Turbo Professionnel 5.0

Bibliothèque de routines optimisées pour Turbo Pascal

Volume 2

Ce logiciel et la documentation qui l'accompagne bénéficient de la protection légale accordée aux droits d'auteur. Toute utilisation de ce logiciel suppose que vous (ou votre employeur) ayez conclu un contrat de licence. Pour cela, complétez et renvoyez la licence jointe à ce produit.

Toute reproduction, sous quelque forme que ce soit (y compris électronique ou mécanique, photocopie) est strictement interdite sans l'accord préalable et écrit de INFOLYS-ATEA.

Les informations contenues dans ce document sont sujettes à révision sans préavis.

© Copyright 1987-1988

Logiciel et version anglaise : TurboPower Software

Version française : Infolys-Atea

Marques déposées

TurboPower Software et le logo TurboPower sont des marques de TurboPower Software.

ATEA est une marque déposée, exploitée sous licence par Infolys.

Turbo Professional est une marque déposée de Sunny Hill Software, dont l'utilisation est concédée par licence à TurboPower Software.

Turbo Extender et TurboPower Utilities sont des marques de TurboPower Software.

Turbo Pascal et Turbo Database ToolBox, SideKick et SuperKey sont des marques de Borland International.

Hercules est une marque déposée de Hercules Computer Technology, Inc.

Lotus est une marque déposée de Lotus Development Corporation.

Ms-DoS, OS/2, et Macro Assembler sont des marques de Microsoft Corporation.

IBM PC, XT, AT et PS/2 sont des marques déposées de International Business Machines Corporation.

Table des matières

Volume 1

1. Introduction **1.1**
Configuration matérielle 1.2
Contenu du package 1.2
Mise en route 1.4

2. Gestion d'écrans **2.1**
Généralités 2.1
TPCRT : Unité CRT améliorée 2.4
TPFENETR : Fenêtres "popup" 2.81
TPMENU : Menus déroulants 2.117
TPEDIT : Editeur Ligne 2.168
TPAIDE : Aide adaptée au contexte 2.185
TPSOURIS : Routines de gestion de la souris 2.220
TPPICK : Listes Pick 2.262
TPDIR : Répertoires "popup" 2.282
TPECRAN : Ecrans virtuels 2.289

3. Saisie de données **3.1**
Généralités 3.1
TPSAISIE : Saisie de données 3.3
TPDATE : Variables de type Date et de type Heure 3.96

Index des procédures et fonctions **I.1**

Volume 2

4. Manipulation des chaînes **4.1**
Généralités 4.1
TPCHAINE : Manipulation de chaînes de caractères 4.3
TPASCIIZ : Chaînes longues ASCIIZ 4.62
TPCHNFOR : Chaînes formatées 4.92

5. Appels au DOS et au BIOS **5.1**
Généralités 5.1
TPDOS : Appels au DOS ou au BIOS 5.3
TPCLPLUS : Interface de clavier étendu 5.35

6. Interruptions et programmes résidents **6.1**
Généralités 6.1
TPINT : Routines de gestion d'interruption (ISR) 6.4
TPTSR : Gestion des programmes résidents 6.21
TP8087 ..... 6.51
TPINT24 : Gestionnaire d'erreurs critiques 6.58

7. Tri **7.1**
Généralités 7.1
TPTRI : Routine de tri en mémoire 7.4

8. Allocation mémoire **8.1**
Généralités 8.1
TPEMS : Gestion de la mémoire EMS 8.4
TPMEMEXT : Gestion de la mémoire étendue 8.20

9. Macros **9.1**
Généralités 9.1
TPMACRO : Macros clavier 9.3
TPEDMAC : Editeur de macros 9.20

10. Grands tableaux **10.1**
Généralités 10.1
TPTBMRAM : Tableaux en RAM normale 10.4
TPTBMEXP : Tableaux en mémoire EMS 10.20
TPTBMVIR : Tableaux en mémoire virtuelle 10.33

11. Arithmétique BCD **11.1**
Généralités 11.1
TPBCD : Réels en haute précision 11.3

12. Divers **12.1**
Généralités 12.1
TPEREXEC : Gestion des erreurs d'exécution 12.2
TPINLINE : Macros inline 12.7

13. Programmes de démonstration **13.1**
SMACS : Processeur de macros 13.2
CALC : Calculatrice du programmeur 13.6
GESTEMPS : Utilitaire de gestion d'emploi du temps 13.11
GENETAT : Générateur d'états pour GESTEMPS 13.14
REF : Table synoptique des caractères 13.17
CREEMENU : Générateur de menus 13.19
BTRI : Utilitaire de tri de fichier 13.30
CHERCHE : Utilitaire de recherche dans des fichiers textes 13.31
REPETE : Répétiteur de commandes 13.33
DIFF : Recherche des différences 13.36
VBROWSE : "Browse" de fichiers virtuels 13.37
SAISIE : Exemple de saisie de données 13.39
Autres programmes 13.44

14. Annexes **14.1**
Ordre d'écriture des unités 14.1
Les nouveautés de Turbo Professionnel 5.0 ..... 14.4
Gestion du clavier avec Turbo Professionnel 14.6
Utilisation des "overlays" avec Turbo Professionnel 14.16

Index des procédures et fonctions **I.1**

4 Manipulation des chaînes de caractères

Généralités

La plupart des bibliothèques commercialisées contiennent un certain nombre de routines pour la manipulation des chaînes et *Turbo Professionnel* n'y fait pas exception. Comme vous vous y attendez, vous trouverez les routines habituelles pour convertir des nombres en chaînes de caractères avec de nombreux formats, modifier certains caractères d'une chaîne en majuscules ou en minuscules, éliminer les blancs de tête et de queue, compléter les chaînes avec des blancs et effectuer des comparaisons sur les chaînes.

L'unité TPCHAINE est composée principalement de routines de manipulation de chaînes de portée générale, y compris toutes les catégories de routines mentionnées ci-dessus. Elle contient également un certain nombre de routines plus inattendues. Les plus remarquables appartiennent à un ensemble complet de routines pour gérer les noms de chemins et de fichiers DOS. Elles vous permettent d'ajouter et de supprimer des identificateurs de chemins et des suffixes, de corriger des noms de fichiers contenant des caractères supplémentaires et de mettre sous une forme correcte des noms complets de chemins sous la forme Lecteur:\Chemin\NomFichier.Ext. L'unité TPCHAINE comporte aussi des routines pour allouer et libérer facilement de la place sur le tas pour des chaînes de caractères ; pour rechercher dans de grands tableaux des chaînes spécifiées et pour convertir des chaînes de caractères en code *Soundex*, permettant de rechercher des noms qui soient les plus proches possibles tel que "*Mallet*" et "*Mallait*".

L'une des routines les plus pratiques dans l'unité est certainement la fonction *Formate* qui est calquée sur la fonction *Format* dans la version BCD de *Turbo Pascal* 3.0. *Formate* prend deux paramètres, une chaîne de caractères, *Masque*, et un nombre réel et renvoie une chaîne formatée qui fait correspondre les chiffres individuels du nombre réel aux emplacements correspondants dans le masque.

Par exemple, étant donné le masque "$##,###,###.##" et le nombre réel 12300.26666, *Formate* renverra " $12,300.27" (remarquez qu'il existe une seconde version de la fonction *Formate* dans l'unité TPBCD, réservée au traitement des nombres réels en haute précision BCD).

Un des reproches les plus classiques fait à *Turbo Pascal* est que ses chaînes de caractères sont limitées à 255 caractères. Bien que 255 caractères soient en général suffisant, lorsque vous avez réellement besoin de chaînes plus longues, vous êtes dans l'embarras. L'unité TPASCIIZ fournit une solution à ce problème. Contrairement aux chaînes habituelles de Turbo, une chaîne ASCIIZ ne contient pas d'octet de longueur. La fin de la chaîne est signalée par un seul caractère nul (#0, le caractère Z de ASCIIZ).

L'inconvénient majeur de l'ASCIIZ est que certaines opérations telles que la recherche de la longueur d'une chaîne peut prendre beaucoup de temps mais l'énorme avantage est qu'une chaîne peut être beaucoup plus longue (au plus 65520 caractères). Naturellement, l'autre inconvénient de ASCIIZ est que le compilateur Turbo ne peut pas supporter ces chaînes mais TPASCIIZ contient ce qu'il faut pour travailler avec elles. En plus des routines de manipulations de chaînes propres à Turbo telles que *Concat, Insert, Delete, Pos* et *Length* --TPASCIIZ contient la plupart des routines de fonctions équivalentes dans TPCHAINE, y compris les routines pour les conversions de majuscules, minuscules, de comparaisons de chaînes, d'élimination des blancs, de compléments avec des blancs et d'allocation dynamique des variables ASCIIZ. Enfin, elle contient des routines pour lire, écrire des chaînes de caractères sur un fichier texte ou à partir d'un fichier texte.

La petite unité TPCHNFOR a une seule fonction : elle permet d'utiliser la puissance de l'instruction *Write* pour créer des chaînes de caractères formatées sans avoir à les envoyer sur un fichier. Cette possibilité est particulièrement utile si vous utilisez la routine *EcritVite* de TPCRT qui n'accepte pas un nombre comme paramètre et qui ne complète pas les chaînes de caractères avec des blancs. Par exemple, si vous voulez convertir les entiers -357 et 41 en une seule chaîne de 18 caractères, vous pourriez utiliser les fonctions *ChnEnEntier* et *Complete* de TPCHAINE puis envoyer le résultat de ces différents appels à la fonction *EcritVite*. Toutefois, vous pourriez le faire plus simplement en utilisant l'unité TPCHNFOR.

```
Write(TpChn, -357, 41:14);
EcritVite(RetourneChn, Ligne, Colonne, Attribut);
```

Plus généralement, TPCHNFOR permet d'utiliser les formats des instructions Write si vous n'écrivez pas sur l'écran, ou sur un fichier texte.

TPCHAINE : Manipulation de chaînes de caractères

TPCHAINE contient toutes les routines de manipulation de chaînes que vous pouvez désirer et un certain nombre que vous n'attendiez pas. Nous pensons qu'elle contient ce qu'il faut pour un programme professionnel qui manipule des chaînes de caractères. TPCHAINE travaille avec des chaînes standard de 255 caractères. Si vous avez besoin de chaînes plus longues, consultez le chapitre TPASCIIZ.

Les routines de TPCHAINE sont divisées en plusieurs catégories : formatage et conversion numérique, manipulation de portée générale telle que la conversion de majuscules en minuscules et vice et versa, le complément de chaînes de caractères, l'élimination de caractères et le centrage, les comparaisons et les recherches de chaînes de caractères, l'enregistrement dynamique de chaînes sur le tas et l'analyse des noms de chemins DOS.

Types

```
TableB = array[0..255] of Byte;
```

Table utilisée par les routines de recherche de Boyer-Moore.

```
EnsembleCarac = set of Char;
```

Ensemble de caractères.

```
TypeComparaison = (Inferieur, Egal, Superieur);
```

Résultat de la comparaison de deux variables.

```
{$IFOPT N+}
Float = Extended;
{$Else}
Float = Real;
{$ENDIF}
```

Si TPCHAINE est compilé avec $N+, Float est de type Extended plutôt que de type Real. La définition de Float touche les routines *ReelEnChn*, et *ChnEnReel*.

Constantes

```
VirgulePourPoint : Boolean = False;
```

Si cette constante vaut "True", la fonction *Formate* renverra '.' pour ',' et réciproquement. Cela facilite l'écriture des programmes utilisés à la fois aux Etats-Unis et dans les pays qui utilisent la virgule à la place du point décimal.

```
ValeurLettres : array['A'..'Z'] of byte =
(3 {A},6 {B},5 {C}, 4 {D}, 3 {E},5 {F},5 {G}, 4 {H}, 3 {I},
 8 {J}, 7 {K}, 4 {L}, 5 {M}, 3 {N}, 3 {O}, 5 {P}, 7 {Q}, 4 {R},
 3 {S}, 3 {T}, 4 {U}, 6 {V}, 5 {W}, 8 {X}, 8 {Y}, 9 {Z});
```

Table utilisée par *CompareEnsLettres* pour calculer le degré de voisinage des mots.

```
SigneMonetaire : Char = '$';
```

Ce caractère est utilisé pour afficher des données avec le format $ et cents. Vous pouvez modifier cette constante et lui donner par exemple la valeur #156, qui correspond au symbole de la livre britannique.

AjoutBackSlash

Déclaration

```
function AjoutBackSlash(NomChemin : string) : string;
```

Objet

Ajoute un backslash par défaut à un nom de répertoire.

Commentaires

Le caractère Backslash '\' sépare un nom de répertoire MS-DOS d'un nom de fichier contenu dans ce répertoire. Un cas spécial interdit la simple concaténation d'un backslash à tous les noms de répertoire. Le nom du répertoire racine est composé réellement d'un backslash et le DOS n'aimera pas qu'un backslash supplémentaire soit ajouté. La fonction *AjoutBackSlash* concatène un backslash à un nom de répertoire seulement si le nom de répertoire ne se termine pas par un backslash.

Exemples

```
NomChemin := AjoutBackSlash('C:\'+'AUTOEXEC.BAT');
```

Le nom de chemin de retourné est "C:\AUTOEXEC.BAT"

```
NomChemin := AjoutBackSlash('\TURBO\TEST\'+'BUG.PAS');
```

Le nom de chemin retourné est '\turbo\test\BUG.PAS'

Voir aussi

NomCheminComplet (TPCHAINE)

ArrangeMots

Déclaration

```
procedure ArrangeMots(ChnEnt : string; var ChnSort, Deborde : string;
                      Marge : Byte; CompleteMarge : Boolean);
```

Objet

Cadre *ChnEnt* à la marge et enregistre le résultat dans *ChnSort* et le reste dans *Deborde*.

Commentaires

Cette routine prend une chaîne d'entrée (*ChnEnt*) et renvoie une chaîne de sortie (*ChnSort*) de longueur inférieure à *Marge* caractères. Si la marge se situe au milieu d'un mot (défini comme une suite de caractères terminée par des espaces ou une fin de chaîne), la chaîne de sortie contiendra tous les mots précédents de la chaîne, tandis que le texte compris entre le mot qui déborde et la fin de la chaîne sera renvoyé dans *Deborde*. *ChnSort* doit être suffisamment grande pour contenir une chaîne de *Marge* caractères et *Deborde* doit être supérieure ou égale à *ChnEnt*.

Si la chaîne d'entrée ne peut être cadrée sur des mots, c'est à dire si *Length(ChnEnt)* est supérieure à *Marge* et qu'il n'y a pas d'espaces entre *ChnEnt[1]* et *ChnEnt[Marge]*, la chaîne de sortie contiendra simplement les premiers caractères de marge dans *ChnEnt*.

CompleteMarge indique si la chaîne de sortie doit être complétée ou non avec des blancs à partir de la marge spécifiée.

Exemple

```
var
   ChaineEntree, ChaineSortie : string;
begin
   while not EOF(Input) do begin
      ReadLn(Input, ChaineEntree);
      ChaineEntree := SupTab(ChaineEntree, 8);
      repeat
         ArrangeMots(ChaineEntree,ChaineSortie, ChaineEntree, 40,
                     False);
         WriteLn(Output,ChaineSortie);
      until Length(ChaineEntree) = 0;
   end;
end.
```

Un programme simple de filtrage de fichiers ASCII qui étend toutes les tabulations des lignes d'entrée et cadre les lignes de sortie en colonne 40.

Voir aussi

SupTab(TPCHAINE)

Centre

Déclaration

```
function Centre(S : string; Largeur : Byte) : string;
```

Objet

Renvoie une chaîne centrée dans une chaîne de longueur spécifiée ne contenant que des espaces.

Commentaires

Si la longueur de *S* est supérieure à *Largeur*, *S* est renvoyée inchangée. Si *S* est une chaîne vide, une chaîne de blancs est renvoyée.

Exemple

```
Centre('Bonjour', 10);
```

Retourne la chaîne ' Bonjour '

Voir aussi

CentreCar (TPCHAINE)

CentreCar

Déclaration

```
function CentreCar(S : string; Car : Char; Largeur : Byte) : string;
```

Objet

Renvoie une chaîne centrée dans une chaîne de *Car* de largeur spécifiée.

Commentaires

Si la longueur de *S* est supérieure à *Largeur*, alors *S* est renvoyée inchangée. Si *S* est une chaîne vide, une chaîne de caractères *Car* est renvoyée.

Exemple

```
CentreCar('Bonjour', '-', 10);
```

Retourne '-Bonjour --'

Voir aussi

Centre (TPCHAINE)

ChaineCarac

Déclaration

```
function ChaineCarac(Car : Char; Longueur : Byte) : string;
```

Objet

Renvoie une chaîne composée de *Longueur* caractères *Car*.

Commentaires

Si *Longueur* est égale à zéro, une chaîne vide est renvoyée.

Exemple

```
ChaineCarac('-', 10);
```

Retourne '----------'

Voir aussi

CentreCar (TPCHAINE)

CompleteCar (TPCHAINE)

ChaineDuTas

Déclaration

```
function ChaineDuTas(P : pointer) : string;
```

Objet

Renvoie la chaîne localisée en *P*.

Commentaires

Les applications qui manipulent du texte de façon intensive, sont plus performantes en rangeant les chaînes dans le tas plutôt que comme variables globales. Le tas offre deux avantages dans ce cas : il peut utiliser jusqu'à 640K de mémoire et les chaînes n'utilisent que l'espace dont elles ont besoin plutôt qu'une longueur maximum définie. Supposez, par exemple, qu'un filtre doive lire chaque ligne d'un fichier texte et les stocker en mémoire avant de les traiter. En rangeant chaque ligne dans une donnée globale, un seul caractère utilisera la longueur de ligne maximum. Par contre, en rangeant chaque ligne sur le tas, elle n'utilisera que l'espace dont elle a besoin.

ChaineDuTas fonctionne conjointement avec *ChaineSurTas*. *ChaineSurTas* alloue une chaîne sur le tas et renvoie un pointeur sur cette chaine. *ChaineDuTas* renvoie la chaîne actuellement sur le tas pour une utilisation dans une expression.

Si le pointeur passé à *ChaineDuTas* est à *nil*, la chaîne vide est retournée. *ChaineDuTas* peut être utilisée dans les expressions de chaînes.

Exemple

```
var
   P : Pointer;
   S : string;
begin
   P := ChaineSurTas('FICHPAS.INC');
   S := 'C:\' + ChaineDuTas(P);
end;
```

ChaineSurTas alloue 12 octets sur le tas pour 'FICHPAS.INC' plus l'octet de longueur. *ChaineDuTas* place 'C:\FICHPAS.INC' dans S.

Voir aussi

DeallouеChaine (TPCHAINE) ChaineSurTas (TPCHAINE)

ChaineSurTas

Déclaration

```
function ChaineSurTas(S : string) : Pointer;
```

Objet

Alloue de l'espace pour S et renvoie un pointeur.

Commentaires

Les applications qui traitent du texte de façon intensive, sont plus performantes en rangeant les chaînes dans le tas plutôt que comme variables globales. Le tas offre deux avantages dans ce cas : il peut utiliser jusqu'à 640K de mémoire et les chaînes n'utilisent que l'espace dont elles ont besoin plutôt qu'une longueur maximum définie. Supposez, par exemple, qu'un filtre doive lire chaque ligne d'un fichier texte et les stocker en mémoire avant de les traiter. En rangeant chaque ligne dans une donnée globale, un seul caractère utilisera la longueur de ligne maximum. Par contre, en rangeant chaque ligne sur le tas, elle n'utilisera que l'espace dont elle a besoin.

ChaineSurTas fonctionne conjointement avec *ChaineDuTas* et *DeallouéChaine*. *ChaineSurTas* alloue une chaîne sur le tas et renvoie un pointeur sur cette chaîne. *ChaineDuTas* renvoie la chaîne actuellement sur le tas pour un usage ultérieur dans une expression.

S'il n'y a pas assez de place sur le tas pour enregistrer une chaîne, *ChaineSurTas* renvoie un pointeur ayant la valeur nil.

Voir aussi

DeallouéChaine (TPCHAINE) ChaineDuTas (TPCHAINE)

ChnEnEntier

Déclaration

```
function ChnEnEntier(S : string; var I : Integer) : Boolean;
```

Objet

Convertit une chaîne en un entier et renvoie la valeur "True" si l'opération est bien terminée.

Commentaires

Cette routine appelle la procédure interne *Val* de Turbo, affecte le résultat numérique au paramètre VAR et renvoie un code booléen comme résultat de la fonction. Tous les blancs de fin sont supprimés de *S* avant l'appel à la procédure *Val*.

Si *ChnEnEntier* renvoie "False", le paramètre variable *I* contiendra la position dans S qui a entraîné l'erreur. Si la chaîne d'entrée commence par $, la chaîne est interprétée comme un nombre hexadécimal. Les blancs de tête sont autorisés.

Exemples

```
ChnEnEntier('  123',I);
```

Retourne "True" et I = 123

```
ChnEnEntier('$00F0',I);
```

Retourne "True" et I = 240

```
ChnEnEntier('12z',I);
```

Retourne "False" et I = 3

ChnEnLongEntier

Déclaration

```
function ChnEnLongEntier(S : string; var I : LongInt) : Boolean;
```

Objet

Convertit une chaîne en un entier long et renvoie la valeur "True" si l'opération est bien terminée.

Commentaires

Cette routine appelle la procédure interne *Val* de Turbo et affecte un résultat numérique au paramètre VAR et un code booléen en résultat à la fonction. *Val.* Tous les blancs de fin sont supprimés de *S* avant l'appel à la procédure *Val.*

Si *ChnEnLongEntier* renvoie "False", le paramètre variable *I* contiendra la position dans *S* qui a entraîné l'erreur. Si la chaîne d'entrée commence par $, la chaîne est interprété comme un nombre hexadécimal, les blancs de tête sont autorisés.

Exemple

```
ChnEnLongEntier('$A0000', I);
```

Retourne "True" et I = 655360

Voir aussi

ChnEnEntier (TPCHAINE)

ChnEnMot

Déclaration

```
function ChnEnMot(S : string; var I : Word) : Boolean;
```

Objet

Convertit une chaîne en un mot et renvoie la valeur "True" si l'opération est bien terminée.

Commentaires

Cette routine appelle la procédure interne *Val* de Turbo et affecte un résultat numérique au paramètre VAR et un code booléen au résultat de la fonction. Tous les blancs de fin sont supprimés de *S* avant l'appel à la procédure *Val*.

Si *ChnEnMot* renvoie "False", le paramètre *I* contiendra la position dans *S* qui a entraîné l'erreur. Si la chaîne d'entrée commence par $, la chaîne est interprétée comme un nombre hexadécimal. Les blancs de tête sont autorisés.

Exemples

```
ChnEnMot('  123', I);
```

Retourne "True" et I = 123

```
ChnEnMot('12z', I);
```

Retourne "False" et I = 3

Voir aussi

ChnEnEntier (TPCHAINE)

ChnEnReel

Déclaration

```
function ChnEnReel(S : string; var R : Float) : Boolean;
```

Objet

Convertit une chaîne de caractères en un réel et renvoie "True" en cas de succès.

Commentaires

Cette routine appelle la procédure interne *Val* de Turbo, qui renvoie le résultat numérique dans le paramètre VAR et un code booléen indiquant si la fonction s'est bien terminée. Les blancs de queue sont éliminés avant l'appel à la routine *Val*. Sous Turbo 4.0, *ChnEnReel* accepte des nombres tels que '5', '3', et '-.1' comme paramètre de *Val*. Si *ChnEnReel* renvoie "False", le paramètre par variable R contiendra la position dans *S* où l'erreur s'est produite.

Remarque : Si TPCHAINE est compilée avec $N+, cette routine fonctionne avec des variables de type Extended, sinon avec des variables de type Real.

Exemple

```
ChnEnReel(' -.123 ', R);
```

Renvoie "True" et R = -0.123.

```
ChnEnReel('123. 4567', R);
```

Renvoie "False" et R = 5.0.

ChnMajusc

Déclaration

```
function ChnMajusc(S : string) : string;
```

Objet

Convertit les lettres minuscules de la chaîne en majuscules.

Commentaires

Notez que *ChnMajusc* gère correctement tous les caractères alphabétiques du jeu de caractère international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemples

```
ChnMajusc('bonjour');
```

Renvoie 'BONJOUR'.

```
ChnMajusc('&#@ &!..\abcd');
```

Renvoie '&#@ &!..\ABCD'

Voir aussi

ChnMinusc (TPCHAINE) UpCase(TPCHAINE)

ChnMinusc

Déclaration

```
function ChnMinusc(S : string) : string;
```

Objet

Convertit les lettres majuscules de la chaîne en minuscules. Notez que *ChnMinusc* gère correctement tous les caractères alphabétiques du jeu de caractère international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemples

```
ChnMinusc('ABC');
```

Renvoie 'abc'.

```
ChnMinusc('!@# ..\abcd!');
```

Renvoie '!@# ..\abcd!'

Voir aussi

Minuscule (TPCHAINE) ChnMajusc (TPCHAINE)

CompareChaine

Déclaration

```
function CompareChaine(S1, S2 : string) : TypeCompare ;
```

Objet

Renvoie "*Inferieur*", "*Egal*", "*Superieur*" selon que s1<s2, s1=s2, ou s1>s2.

Commentaires

CompareChaine fait la même chose que les opérateurs de comparaison de chaînes de Turbo Pascal avec un avantage. Dans un branchement dépendant de trois conditions "*Egal*", "*Superieur*", ou "*Inferieur*", *CompareChaine* donnera le résultat en une seule comparaison tandis que les opérateurs infixés demanderont au moins deux comparaisons. Ainsi, dans certains programmes, *CompareChaine* sera plus rapide.

Exemples

```
var
   L1, L2 : String;
   ....
case CompareChaine(L1,L2) of
   Inferieur    : Writeln('L1 < L2');
   Egal         : Writeln('L1 = L2');
   Superieur    : Writeln('L1 > L2');
end;
```

Cette structure générale peut être utilisée lorsque trois sorties différentes sont rattachées à une seule comparaison de chaînes.

```
CompareChaine('aaa', 'aaaa');
```

Renvoie "*Inferieur*".

```
CompareChaine('aaab', 'aaaa');
```

Renvoie "*Superieur*".

```
CompareChaine('Aaaa', 'aaaa');
```

Renvoie "*Inferieur*".

Voir aussi

CompareChnMajusc (TPCHAINE)

CompareChnMajusc

Déclaration

```
function CompareChnMajusc(S1, S2 : string) : TypeCompare;
```

Objet

Compare deux chaînes de caractères sans tenir compte des majuscules/minuscules.

Commentaires

CompareChnMajusc convertit les minuscules en majuscules avant de comparer les deux chaînes. Bien que plus lente que *CompareChaine* ou que la fonction de comparaison interne de Turbo, *CompareChnMajusc* est plus rapide que l'appel de *ChnMajusc* et que la comparaison normale de chaînes de caractères. *CompareChnMajusc* n'effectue aucun changement sur les chaînes entrées. Notez que *CompareChnMajusc* gère correctement tous les caractères alphétiques du jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemples

```
CompareChnMajusc('AAA','aaaa');
```

Retourne "Inferieur".

```
CompareChnMajusc('AAAB', 'aaaa');
```

Retourne "Superieur".

```
CompareChnMajusc('Aaaa', 'aaaa');
```

Retourne "Egal".

Voir aussi

CompareChaine (TPCHAINE)

CompareEnsLettres

Déclaration

```
function CompareEnsLettres(Ensemble1, Ensemble2 : LongInt) : Word;
```

Objet

Renvoie la somme des valeurs des lettres communes à Ensemble1 et Ensemble2.

Commentaires

Lorsqu'elle est utilisée avec *CreeEnsLettres*, *CompareEnsLettres* est une alternative à l'algorithme Soundex pour définir le degré de voisinage entre deux mots ou chaînes de mots. L'algorithme a été développé par Mickaël Bickel et est expliqué en détails dans "*A Alternative to Soundex*" de Jim Howell (Doctor Dobb's Journal, Novembre 1987).

Avant d'appeler *CompareEnsLettres*, vous devrez utiliser *CreeEnsLettres* pour créer un entier long représenté sous forme binaire (ou "ensemble lettres") contenant les lettres utilisées dans une chaîne donnée; cela devra être répété pour chaque comparaison de chaînes. Etant donné ces deux ensembles, *CompareEnsLettre* calculera l'union des deux ensembles et renverra une valeur indiquant le degré de voisinage des deux chaînes. En calculant cette valeur, il utilisera la table (déclarée dans TPCHAINE.PAS) qui affecte des fréquences relatives élevées aux lettres les moins utilisées ('J', 'Q', 'X', 'Z') et des fréquences faibles aux lettres les plus utilisées ('A', 'E', 'N', 'S', 'T'). La valeur renvoyée est simplement la somme des valeurs de chaque ensemble de l'opération d'union. Pour déterminer la valeur maximum possible, vous pourrez comparer l'ensemble des lettres d'une chaîne avec elle-même.

```
ValeurPossible := CompareEnsLettres(EnsLettres1, EnsLettres1) ;
```

Les comparaisons d'ensembles de lettres pour d'autres chaînes donneront un résultat appartenant à l'intervalle 0..Valeur Possible. Plus la valeur est grande, plus le degré de voisinage est important.

Malheureusement, il n'y a pas de méthode radicale pour décider si la valeur renvoyée par *CompareEnsLettres* correspond au degré de voisinage le plus grand. Aussi vous devrez faire quelques essais pour définir la meilleure utilisation de ces fonctions. Le moyen d'améliorer ces résultats, est d'utiliser une fonction telle que *IndexDifference* comme dans l'exemple ci-dessous qui renvoie une valeur comprise entre 0.0 (voisinage le plus haut) et 130.0 (la première chaîne est vide). (Des chaînes voisines ont habituellement un index de 0.3 ou moins).

On peut aussi modifier la fonction pour prendre en compte des facteurs que l'algorithme de Bickel ignore (est-ce-que les deux chaînes ont la même longueur ? Est-ce-que les premiers caractères sont les mêmes ? etc... Pour des explications plus complètes sur l'algorithme, voir l'article de Howell dans Docteur Dobb).

Exemple

```
function IndexDifference(S1, S2 : String) : Real;
var
   Ens1, Ens2 : LongInt;
   Possible, Actuel, Delta : Integer;
begin
   Ens1 := CreeEnsLettres(S1);
   Ens2 := CreeEnsLettres(S2);
   Possible := CompareEnsLettres(Ens1, Ens1);
   Actuel:= CompareEnsLettres(Ens1, Ens2);
   Delta := Possible - Actuel;
   if Possible = 0 then
      IndexDifference := 130
   else
      IndexDifference := Abs(Delta / Possible);
end;

var S1, S2 : String;
begin
   repeat
      {Lecture de la première chaîne}
      Write('Chaîne 1 : '); ReadLn(S1);
      {Lecture de la deuxième chaîne}
      Write('Chaîne 2 : '); ReadLn(S2);
      {Affichage du résultat}
      WriteLn('Index : ', IndexDifference(S1, S2):1:3);
   until False; {Presser ^C pour arrêter}
end.
```

Voir aussi

CreeEnsLettres(TPCHAINE) Soundex(TPCHAINE)

CompareStruct

Déclaration

```
function CompareStruct(var S1, S2; Taille : Word) : TypeCompare;
```

Objet

Compare deux structures de tailles fixes.

Commentaires

Pascal ne fournit pas de fonctions internes pour comparer des types de données structurés tels que des tableaux ou des enregistrements. Dans certains cas, cette absence est gênante. Par exemple, considérons un enregistrement contenant un entier suivi par une chaîne de caractères. Une routine de comparaison aurait des difficultés à analyser cette structure et à effectuer les comparaisons sur les différents composants.

La fonction *CompareStruct* fournit une solution. Bien que *CompareStruct* renvoie un résultat de type énuméré, *Inferieur, Egal, Superieur,* dans la plupart des cas, seule l'égalité ou la différence seront significatives. *CompareStruct* effectue une comparaison sur les caractères ASCII octet par octet pour déterminer *Inferieur* ou *Egal.*

Remarquez qu'il est de la responsabilité du programmeur d'éviter des chaînes de caractères imbriquées. Le paramètre *Taille* de *CompareStruct* est le nombre d'octets à comparer. La meilleure façon de déterminer ce paramètre est d'utiliser la fonction *SizeOf()* de Turbo.

Exemples

```
type
   Tableau = array[1..3] of byte;
const
   T1 : Tableau = (0, 1, 2);
   T2 : Tableau = (0, 1, 3);
   T3 : Tableau = (0, 1, 2);
...
CompareStruct(T1, T2, SizeOf(Tableau));
```

Retourne "*Inferieur*".

```
CompareStruct(T1, T3, SizeOf(Tableau));
```

Retourne "*Egal*".

Voir aussi

CompareChaine (TPCHAINE)

Complete

Déclaration

```
function Complete(S : string; Longueur : Byte) : string;
```

Objet

Renvoie une chaîne de caractères complétée à droite avec des blancs sur une longueur égale à *Longueur*.

Commentaires

Des blancs sont ajoutés à la fin de la chaîne d'entrée à concurrence de la longueur *Longueur*. Si la longueur de *S* est déjà égale ou supérieure à *Longueur*, *S* est retournée inchangée. Si *S* est une chaîne vide, une chaîne de blancs est renvoyée.

Exemples

```
Complete('Bonjour',10);
```

Retourne 'Bonjour '

```
Complete('gigantesque',10);
```

Retourne 'gigantesque'

Voir aussi

Centre (TPCHAINE)
CompleteCar (TPCHAINE)
CompleteGauche (TPCHAINE)

CompleteCar

Déclaration

```
function CompleteCar(S : string; Car : Char; Longueur : Byte): string;
```

Objet

Renvoie une chaîne cadrée à droite complétée avec *Car* sur une longueur égale à *Longueur*.

Commentaires

Des caractères sont ajoutés à la fin de la chaîne d'entrée à concurrence de la longueur égale à *Longueur*. Si la longueur de *S* est déjà égale ou supérieure à *Longueur*, *S* est renvoyée inchangée. Si *S* est une chaîne vide, une chaîne de *Car* est renvoyée.

Exemples

```
CompleteCar('Bonjour','-',10);
```

Retourne 'Bonjour---'.

```
CompleteCar('gigantesque','-',10);
```

Retourne 'gigantesque'.

CompleteGauche

Déclaration

```
function CompleteGauche(S : string; Long : Byte) : string;
```

Objet

Renvoie une chaîne de caractères complétée à gauche avec des blancs sur une longueur égale à *Long*.

Commentaires

Les blancs sont insérés à gauche de la chaîne d'entrée à concurrence de la longueur spécifiée. Si *S* est plus long que *Long*, alors *S* est inchangé. Si *S* est la chaîne vide, une chaîne de blancs est renvoyée.

Exemples

```
CompleteGauche('Bonjour',10);
```

Retourne ' Bonjour'

```
CompleteGauche('gigantesque',5);
```

Retourne 'gigantesque'

Voir aussi

Centre (TPCHAINE)
Complete (TPCHAINE)
CompleteGaucheCar (TPCHAINE)

CompleteGaucheCar

Déclaration

```
function CompleteGaucheCar(S : string; Car : Char; Longueur : Byte):
string;
```

Objet

Renvoie une chaîne de caractères complétée à gauche avec *Car* sur une longueur égale à *Longueur*.

Commentaires

Les caractères sont insérés à gauche de la chaîne d'entrée à concurrence de la longueur égale à *Longueur*. Si *S* est plus longue que *Longueur*, *S* est retournée inchangée. Si *S* est une chaîne vide, une chaîne de *Car* est retournée.

Exemples

```
CompleteGaucheCar('bonjour', '-', 10);
```

Retourne '---Bonjour'

```
CompleteGaucheCar('gigantesque','-', 4);
```

Retourne 'gigantesque'

Voir aussi

Centre (TPCHAINE)
Complete (TPCHAINE)
CompleteGauche (TPCHAINE)

CompteMots

Déclaration

```
function CompteMots(S : string; DelimsMots : EnsembleCar) : Byte;
```

Objet

Etant donné un ensemble de délimiteurs de mots, renvoie le nombre de mots de *S*.

Commentaires

Cette routine renvoie le nombre de mots de la chaîne spécifiée. *DelimsMots* est un ensemble de délimiteurs de mots utilisé pour définir où commencent et où se terminent les mots. Cet ensemble devrait contenir au moins ' ', mais dans certains programmes vous pourrez utiliser un ensemble plus important tel que celui de l'exemple ci-dessous.

Exemple

```
var
   S : string;
const
   DelimsMots : EnsembleCarac =
      [#0..'"', '.', ',', ':', '?', '(', ')', '[', ']', '{', '}'];
   N : Word = 0;
begin
   while not EOF(Input) do begin
      ReadLn(Input, S);
      Inc(N, CompteMots(S, DelimsMots));
   end;
   WriteLn(N, ' mots dans le fichier en entrée.');
end.
```

Programme simple qui compte les mots lus sur l'unité standard d'entrée.

Voir aussi

ExtraitMot(TPCHAINE)

ConstruitTableBM

Déclaration

```
procedure ConstruitTableBM(ChaineRech : String ; var TB : TableB);
```

Objet

Construit une table de Boyer-Moore.

Commentaires

L'algorithme de Booyer-Moore est une méthode très rapide de recherche d'une chaîne de caractères. La rapidité vient du fait qu'il n'est pas nécessaire de comparer tous les éléments de la chaîne. Les éléments à ignorer sont définis par la table qui doit être initialisée avant la recherche.

La table est un tableau [0..255] d'octets. Pour chaque caractère du code ASCII (0..255), la table contient le nombre de positions à ignorer. La procédure *ConstruitTableBM*, construit la table à partir de la chaîne de caractères à rechercher, *ChaineRech*. La table sera ensuite passée à la routine de recherche de Boyer-Moore.

Les performances seront améliorées si la chaîne à rechercher est courte. En général, vous devrez minimiser le nombre d'appels à *ConstruitTableBM*. Si vous devez appeler cette procédure souvent, il sera préférable d'utiliser les fonctions de recherches classiques.

Voir la fonction *RechercheBM* en exemple.

Voir aussi

RechercheBM(TPCHAINE) RechBMSansMajMin(TPCHAINE)

CorrigeNomChemin

Déclaration

```
function CorrigeNomChemin(NomChemin : string) : string;
```

Objet

Renvoie un nom de chemin corrigé selon la syntaxe du DOS.

Commentaires

Vous avez peut-être remarqué que le DOS accepte les noms de fichiers contenant plus de huit caractères et des suffixes de plus de trois caractères. Il élimine simplement les caractères superflus. Le même comportement s'applique aux noms de sous-répertoires spécifiés séparément ou faisant partie d'un nom de fichier.

Si votre application affiche des noms de fichiers, il serait maladroit d'afficher des caractères ignorés par le DOS. *CorrigeNomChemin* élimine automatiquement tous les caractères superflus.

Exemples

```
CorrigeNomChemin('AUTOEXEC.BAT');
```

Retourne 'AUTOEXEC.BAT'

```
CorrigeNomChemin('AUTOEXECA.BAT1');
```

Retourne 'AUTOEXEC.BAT'

```
CorrigeNomChemin('D:\TEST12345\BUG.PAS');
```

Retourne 'D:\TEST1234\BUG.PAS'

Voir aussi

NomCheminComplet (TPCHAINE)

CreeEnsLettres

Déclaration

```
function CreeEnsLettres(S : string) : LongInt;
```

Objet

Renvoie un entier long binaire qui contient chaque lettre de *S*.

Commentaires

CreeEnsLettres crée un ensemble binaire (sous la forme d'un entier long) qui indique quelles lettres de l'alphabet sont dans *S*. Par exemple, si la chaîne était 'A' (ou 'a'), *CreeEnsLettres* renverrait $00000001 ; si la chaîne était 'ABC', le résultat serait $00000007, et ainsi de suite. La valeur maximum possible pour le résultat est $003FFFFF (les 26 premiers bits à 1). Voir les explications de *CompareEnsLettres* pour plus de détails.

Voir aussi

CompareEnsLettres(TPCHAINE)

DeallouеChaine

Déclaration

```
procedure DeallouеChaine(P : Pointer);
```

Objet

Libère l'espace utilisé par une chaîne.

Commentaires

Pour les applications qui traitent de gros volumes de texte, il est préférable d'enregistrer les chaînes sur le tas plutôt que comme des variables globales. Le tas offre deux avantages dans ce cas : il peut s'étendre jusqu'à 640 K dans la mémoire DOS et les chaînes peuvent utiliser uniquement la mémoire dont elles ont besoin plutôt que la longueur maximum possible. Par exemple, supposez qu'un filtre doive lire chaque ligne d'un fichier texte et l'enregistrer en mémoire avant de la traiter. En enregistrant chaque ligne comme une donnée globale, même un seul caractère occuperait la longueur de ligne maximum. En enregistrant les lignes sur le tas, chacune d'elles occupera exactement la place dont elle a besoin.

DeallouеChaine fonctionne conjointement avec *ChaineSurTas*. *ChaineSurTas* alloue de l'espace sur le tas pour une chaîne spécifiée et renvoie un pointeur sur la chaîne. *DeallouеChaine* libère l'espace sur le tas associé à ce pointeur.

Si le pointeur passé à *DeallouеChaine* vaut nil, la fonction ne fait rien.

Exemple

```
var
   P : Pointer;
   S : string;
begin
   P := ChaineSurTas('Bonjour');
   S := ChaineDuTas(P);
   DeallouеChaine(P);
end.
```

L'instruction *ChaineSurTas* alloue huit octets sur le tas pour 'Bonjour' et l'octet de longueur. *ChaineDuTas* charge 'Bonjour' dans *S*, *DeallouеChaine* libère l'espace initial de huit octets sur le tas et affecte la valeur nil au pointeur.

Voir aussi

ChaineDuTas (TPCHAINE) ChaineSurTas (TPCHAINE)

EffBlancs

Déclaration

```
function EffBlancs( S : string) : string;
```

Objet

Renvoie une chaîne dont les espaces de tête et de queue ont été éliminés.

Commentaires

Le caractère espace est ici défini comme n'importe quel caractère dont le rang ASCII est inférieur ou égal 32. Les espaces internes de la chaîne *S* sont inchangés. Si la chaîne d'entrée est composée entièrement d'espaces, la chaîne vide est renvoyée.

Exemple

```
EffBlancs('  Bonjour    ');
EffBlancs(^I^I'Bonjour'^M^J);
```

Retournent 'Bonjour'.

Voir aussi

EffEspDeb (TPCHAINE) EffEspFin (TPCHAINE)

EffEspDeb

Déclaration

```
function EffEspDeb(S : string) : string;
```

Objet

Renvoie une chaîne dont les espaces de tête ont été éliminés.

Commentaires

Le caractère espace est ici défini comme n'importe quel caractère dont le rang ASCII est inférieur ou égal 32. Les espaces internes de la chaîne *S* sont inchangés. Si la chaîne d'entrée est composée entièrement d'espaces, la chaîne vide est renvoyée.

Exemples

```
EffEspDeb('   Bonjour   ');
```

Renvoie 'Bonjour '

```
EffEspDeb(^I^I'Bonjour'^M^J);
```

Renvoie 'Bonjour'^M^J

Voir aussi

EffBlancs (TPCHAINE)

EffEspFin (TPCHAINE)

EffEspFin

Déclaration

```
function EffEspFin(S : string) : string;
```

Objet

Renvoie une chaîne dont les espaces situés à la fin ont été éliminés.

Commentaires

Le caractère espace est ici défini comme n'importe quel caractère ASCII <=32.

Les espaces internes de la chaîne *S* sont inchangés. Si la chaîne d'entrée est composée entièrement d'espaces, la chaîne vide est renvoyée.

Exemples

```
EffEspFin('   Bonjour   ');
```

Renvoie ' Bonjour'

```
EffEspFin(^I^I'Bonjour'^M^J);
```

Renvoie ^I^I'Bonjour'

Voir aussi

EffBlancs (TPCHAINE)

EffEspDeb (TPCHAINE)

ExtraitMot

Déclaration

```
function ExtraitMot(N : Byte; S : string; DelimsMot : EnsembleCar):
string;
```

Objet

Etant donné un jeu de délimiteurs de mots renvoie le Nième mot de C.

Commentaires

Cette routine renvoie le Nième mot de la chaîne spécifiée. *DelimsMot* est un ensemble de délimiteurs utilisés pour déterminer où commencent et où finissent les mots. Cet ensemble devrait contenir au moins ' '. Si *S* contient moins de *N* mots, la chaîne renvoyée sera vide.

Exemple

```
S := 'Il y a quatre vingt sept ans, nos aïeux...';
SeptiemeMot := ExtraitMot(7, S, [' ', ',', '.']);
```

Le résultat sera 'ans', puisque ',' est le délimiteur de mots.

Voir aussi

CompteMots(TPCHAINE).

ForceSuffixe

Déclaration

```
function ForceSuffixe(Nom, Suffixe : string) : string;
```

Objet

Ajoute un suffixe à un nom de fichier.

Commentaires

Que le nom ait ou n'ait pas de suffixe, *ForceSuffixe* renvoie un nouveau nom de fichier avec le suffixe spécifié.

Exemples

```
ForceSuffixe('TEST','pas');
```

Retourne 'TEST.pas'

```
ForceSuffixe('TEST','PAS');
```

Retourne 'TEST.PAS'.

```
ForceSuffixe('C:\AUTOEXEC.BAT','PAS');
```

Retourne 'C:\AUTOEXEC.PAS'.

Voir aussi

SuffixeParDefaut (TPCHAINE)

Formate

Déclaration

```
function Formate(Masque : string; R : Float) : string;
```

Objet

Renvoie une chaîne de caractères formatées en appliquant un masque sur la donnée *R*.

Commentaires

Formate fournit l'équivalent de la fonction *Form* de Turbo-BCD pour les réels sur 6 octets ou réels 8087 de la version 4.0 de *Turbo Pascal*. La fonction *Formate* offre des possibilités de formatage numérique avancé utilisables principalement dans les calculs financiers.

Puisque Pascal ne permet pas aux routines utilisateurs d'accepter un nombre variable de paramètres, la fonction *Formate* de *Turbo Professionnel* accepte seulement un seul paramètre de type réel.

Le paramètre *Masque* est une chaîne qui peut contenir n'importe quelle combinaison de caractères ainsi qu'une séquence de caractères spéciaux (# * @ $ - + , .). Cette séquence est appelée champ numérique et *Formate* la remplacera par le nombre réel formaté. Le nombre est retourné cadré à droite à l'intérieur du champ, la partie fractionnaire est arrondie si elle dépasse le nombre de positions décimales spécifiées par le format et si le nombre est trop grand pour être retourné dans ce champ. Toutes les positions du *Masque* seront remplies par des astérisques.

Des caractères spéciaux sont interprétés comme suit :

Position de chiffre. Si le champ numérique ne contient pas de caractères '*' ou '@', les chiffres inutilisés sont remplacés par des blancs. Si le champ numérique ne contient pas de position de signe (+ ou -) et que le nombre est négatif, le signe - est placé devant le nombre. Exemples :

```
Formate('####', 34.567);      renvoie '  35'
Formate('####.##', -12.13);   renvoie ' -12.30'
Formate('##.##', 1000.0);     renvoie '**.**'
```

* Position de chiffre. Les positions inutilisées sont renvoyées comme des astérisques à la place de blancs. Le caractère astérisque doit apparaître une seule fois dans le champ numérique pour activer ce mécanisme. Le signe du nombre ne sera pas retourné à moins que le champ contienne un caractère de signe (- ou +). Exemples :

```
Formate('*##.#', 4.567);      renvoie '**4.6'
Formate('****', 123);         renvoie '  *123'
```

@ Position de chiffre. Les positions inutilisées sont retournées comme des zéros plutôt que comme des blancs. Le caractère @ doit apparaître une seule fois dans le champ numérique pour activer ce mécanisme. Le signe du nombre ne sera pas retourné sauf si le champ contient également un signe (- ou +). Exemples :

```
Formate('@##', 9);          renvoie '009'
Formate('@@@.@@', 12.3);    renvoie '012.30'
```

$ Position de chiffre. Un signe $ est renvoyé devant le nombre. Le caractère $ doit apparaître une seule fois dans le champ numérique pour mettre en oeuvre ce mécanisme. Exemples :

```
Formate('$####.##', 123.45); renvoie ' $123.45'
Formate('*$####.##', 12.34); renvoie '***$12.34'
```

- Position de signe. Si le nombre est négatif, un signe - sera renvoyé à cette position. S'il est positif, un blanc sera retourné. Exemples :

```
Formate('-###.##', -1.2);   renvoie '- 1.20'
Formate('-###.##', 1.2);    renvoie '  1.20'
```

+ Position de signe. Si le nombre est positif, un signe + sera renvoyé à cette position. S'il est négatif, un signe - sera renvoyé. Exemples :

```
Formate('+###.##', 1.2);        renvoie '+ 1.20'
Formate('*$####.##+', -12.34); renvoie '***$12.34-'
```

, Virgule décimale ou virgule de séparation.

. Point décimal ou point de séparation. Le dernier point ou la dernière virgule dans un champ numérique est considéré comme le délimiteur décimal. Exemples :

```
Formate('##,###,###.##', 12345678.91); renvoie '12,345,678.91'
Formate('$#.###.###,##', -12345.6);    renvoie '  -$12.345,60'
```

Remarque : Plusieurs utilisateurs des versions précédentes de *Turbo Professionnel*, nous ont fait remarquer qu'ils avaient trouvé un bug dans *Formate*. Considérons ce qui suit :

```
Writeln(Formate('###,###,###', 1000.0));
```

Ce qu'ils attendaient était '1,000', ce qu'ils obtenaient était '1,00,00' ; ce n'est pas un bug. Ceci est compatible avec le fonctionnement de *Form* dans *TurboBcd* 3.0, qui traite le dernier point ou la dernière virgule du masque comme point décimal, permettant à *Form* d'être compatible dans les pays qui utilisent des virgules là où les américains utilisent des points.

Voir aussi

Formate (TPBCD)

LongEnChn

Déclaration

```
function LongEnChn(L : LongInt) : string;
```

Objet

Convertit un entier long, un mot, un entier, un octet, un entier court en une chaîne de caractères.

Commentaires

Cette routine appelle la procédure interne *Str* de Turbo et affecte le résultat à la fonction. La chaîne résultante ne comporte pas de blanc en tête ou à la fin.

Le changement automatique de type de Turbo permet de passer des octets, des entiers ou des mots en paramètres de cette routine.

Exemples

```
LongEnChn(1000);
```

Retourne '1000'.

```
LongEnChn(-1);
```

Retourne '-1'.

Voir aussi

ChnEnEntier (TPCHAINE)

LongEntierBin

Déclaration

```
function LongEntierBin(L : LongInt) : string;
```

Objet

Renvoie la chaîne de bits correspondant à un entier long.

Commentaires

Renvoie une chaîne de 32 caractères qui représentent la valeur binaire d'un entier long. Aucun caractère de séparation n'est inséré dans la chaîne. Les zéros de tête sont inclus.

Exemples

```
LongEntierBin(-1);
```

Retourne '11111111111111111111111111111111'

```
LongEntierBin(7);
```

Retourne '00000000000000000000000000000111'

Voir aussi

OctetBin (TPCHAINE) MotBin (TPCHAINE)

LongEntierHex

Déclaration

```
function LongEntierHex(L : LongInt) : string;
```

Objet

Renvoie la chaîne hexadécimale correspondant à un entier long.

Commentaires

Renvoie une chaîne hexadécimale de huit caractères correspondant à un entier long. Les zéros de tête sont inclus, mais pas les caractères "$" ou "h".

Exemples

```
LongEntierHex(0);
```

Retourne '00000000'

```
LongEntierHex(655360);
```

Retourne '000A0000'

Voir aussi

OctetHex (TPCHAINE) MotHex (TPCHAINE)

LongEntierOctal

Déclaration

```
function LongEntierOctal(L : LongInt) : string;
```

Objet

Renvoie la chaîne octale correspondant à un entier long.

Commentaires

Renvoie une chaîne octale de 12 caractères (base 8). Les zéros de tête sont inclus.

Exemples

```
LongEntierOctal(0);
```

Retourne '000000000000'.

```
LongEntierOctal(655360);
```

Retourne '000002400000'.

Voir aussi

OctetOctal (TPCHAINE)

MotOctal (TPCHAINE)

MacroMajuscule

Déclaration

```
function MacroMajuscule(Car : Char) : Char;
```

Objet

Renvoie *Car* en majuscules.

Commentaires

Macro "inline" qui convertit les caractères en majuscules aussi rapidement que possible.

Exemple

```
MacroMajuscule('a');
```

Renvoie 'A'

```
MacroMajuscule('%');
```

Renvoie '%'

Voir aussi

MacroMinuscule(TPCHAINE)

MacroMinuscule

Déclaration

```
function MacroMinuscule(Car : Char) : Char;
```

Objet

Renvoie *Car* en minuscule.

Commentaires

Macro "inline" qui convertit les caractères minuscules aussi rapidement que possible.

Exemple

```
MacroMinuscule('A');
```

Renvoie 'a'

```
MacroMinuscule('%');
```

Renvoie '%'

Voir aussi

MacroMajuscule(TPCHAINE)

Minuscule

Déclaration

```
function Minuscule(Car : Char) : Char;
```

Objet

Renvoie *Car* en minuscule.

Commentaires

Cette routine convertit les caractères en minuscules. Contrairement à *MacroMinuscule, Minuscule* traite correctement tous les caractères alphabétiques du jeu de caractères internationales ('a'..'z','a'..'z', #128..#154, #160..#167).

Renvoie #164 ('n' avec un tilde).

Exemples

```
Minuscule('A');
```

Retourne 'a'.

```
Minuscule(#165); {'N' et '~'}
```

Retourne '#164' ('n' avec un tilde).

Voir aussi

ChnMinusc(TPCHAINE) UpCase(TPCHAINE)

MotBin

Déclaration

```
function MotBin(W : Word) : string;
```

Objet

Renvoie la chaîne binaire correspondant à un mot.

Commentaires

Renvoie une chaîne de 16 caractères qui représentent la valeur binaire d'un mot. Aucun caractère de séparation n'est inséré dans la chaîne. Les zéros de tête sont inclus.

Exemples

```
MotBin($B800);
```

Retourne '1011100000000000'

```
MotBin(-1);
```

Incorrect ! Ceci ne sera pas compilé car les entiers de type Word sont toujours positifs. Utilisez *LongEntierBin* s'il s'agit d'un nombre inférieur à zéro.

Voir aussi

OctetBin (TPCHAINE)

LongEntierBin (TPCHAINE)

MotHex

Déclaration

```
function MotHex(W : Word) : string;
```

Objet

Renvoie la chaîne hexadécimale correspondant à un mot.

Commentaires

Renvoie 4 caractères hexadécimaux pour le mot. Les zéros de tête sont inclus, les caractères "$" ou "h" ne sont pas inclus.

Exemples

```
MotHex(0);
```

Retourne '0000'.

```
MotHex(MaxInt);
```

Retourne '7FFF'.

Voir aussi

OctetHex (TPCHAINE) LongEntierHex (TPCHAINE)

MotOctal

déclaration

```
function MotOctal(W : Word) :string;
```

Objet

Retourne une chaîne octale correspondant à un mot.

Commentaires

Retourne une chaîne de six caractères en octal (base 8). Les zéros de tête sont inclus.

exemple

```
MotOctal(0)
```

Retourne '000000'.

```
MotOctal(65535)
```

Retourne '177777'.

Voir aussi

OctetOctal (TPCHAINE) LongEntierOctal (TPCHAINE)

NomCheminComplet

déclaration

```
function NomCheminComplet(NomF : string) : string;
```

Objet

Etant donné *NomF* (nom de fichier existant), renvoie un nom de chemin complet.

Commentaires

NomCheminComplet ne recherche pas le PATH du DOS pour retrouver un fichier (voir *ExisteSurChemin* pour cette opération). *NomCheminComplet* renvoie un nom de chemin complet DOS à partir d'un nom qui permet déjà au DOS de retrouver le fichier. Le nom de chemin comprendra une lettre correspondant à un lecteur, un répertoire complet et un nom de fichier.

Dans de nombreux cas, *NomCheminComplet* doit questionner le DOS pour déterminer le répertoire actuel sur le lecteur spécifié ou le lecteur actuel. A cette fin, une disquette doit être chargée dans le lecteur pour éviter un erreur du DOS.

Exemples

```
NomCheminComplet('AUTOEXEC.BAT');
```

Retourne 'C:\AUTOEXEC.BAT' si le lecteur actuel est C:

```
NomCheminComplet('D:..\TEST\VERIF.PAS');
```

Retourne 'D:\EXEMPLES\..\TEST\CHECK.PAS' si le répertoire actuel sur le lecteur D: est \EXEMPLES.

Voir aussi

ExisteSurChemin (TPDOS)

NomCheminSeul

Déclaration

```
function NomCheminSeul(NomChemin : string) : string;
```

Objet

Renvoie simplement le numéro de lecteur et la partie répertoire d'un nom de chemin.

Commentaires

Le nom de fichier, le suffixe de la chaîne d'entrée sont éliminés.

Exemples

```
NomCheminSeul('C:\AUTOEXEC.BAT');
```

Retourne 'C:\'.

```
NomCheminSeul('..\test\bug.pas');
```

Retourne '..\test'.

Voir aussi

SuffixeSeul (TPCHAINE) NomFichSeul (TPCHAINE)

NomFichSeul

Déclaration

```
function NomFichSeul(NomChemin :string) : string;
```

Objet

Renvoie simplement le nom de fichier et le suffixe d'un nom de chemin.

Commentaires

Le numéro de lecteur et le répertoire sont éliminés de la chaîne d'entrée.

Exemples

```
NomFichSeul('C:\AUTOEXEC.BAT');
```

Retourne 'AUTOEXEC.BAT'.

```
NomFichSeul('TEST.');
```

Retourne 'TEST.'.

Voir aussi

SuffixeSeul (TPCHAINE) NomCheminSeul (TPCHAINE)

OctetBin

Déclaration

```
function OctetBin(B : Byte) : string;
```

Objet

Renvoie la chaîne de bits composant un octet.

Commentaires

Renvoie une chaîne de huit caractères qui représente la valeur binaire d'un octet, zéros de tête y compris.

Exemples

```
OctetBin(0);
```

Retourne '00000000'

```
OctetBin($B1);
```

Retourne '10110001'

Voir aussi

LongEntierBin (TPCHAINE) MotBin (TPCHAINE)

OctetHex

Déclaration

```
function OctetHex(B : Byte) : string;
```

Objet

Renvoie la chaîne hexadécimale correspondant à un octet.

Commentaires

Renvoie deux caractères hexadécimaux pour un octet. Les zéros de tête sont inclus. Les caractères '$' ou 'h' ne sont pas inclus.

Exemples

```
OctetHex(0);
```

Retourne '00'.

```
OctetHex(250);
```

Retourne 'FA'.

Voir aussi

LongEntierHex (TPCHAINE) MotHex (TPCHAINE)

OctetOctal

Déclaration

```
function OctetOctal(B : Byte) : string;
```

Objet

Renvoie la chaîne octale correspondant à un octet.

Commentaires

Renvoie une chaîne de trois caractères en octal (base 8). Les zéros de tête sont inclus.

Exemples

```
OctetOctal(0);
```

Retourne '000'

```
OctetOctal(255);
```

Retourne '377'

Voir aussi

LongEntierOctal (TPCHAINE) MotOctal (TPCHAINE)

PlaceTab

Déclaration

```
function PlaceTab(S : String ; TailleTab : Byte) : String;
```

Objet

Convertit les blancs d'une chaîne *S* en tabulations sur une longueur égale à *TailleTab*.

Commentaires

PlaceTab comprime une chaîne en convertissant les espaces en une suite appropriée de tabulations et d'espaces.

PlaceTab reconnaît les chaînes de littéraux délimitées par des apostrophes simples ou des guillemets et ne convertit pas les espaces en tabulations à l'intérieur de ces chaînes. Elle reconnaît aussi le caractère "ESC" '\' pour autoriser des apostrophes à l'intérieur d'une chaîne de caractères.

TailleTab = 8 est la taille normale utilisée par la plupart des éditeurs de textes. Si *TailleTab* vaut 0, *PlaceTab* renverra toujours une chaîne vide.

Exemple

```
var
   S : string;
begin
   while not EOF(Input) do begin
      ReadLn(Input, S);
      WriteLn(OutPut, PlaceTab(S, 8));
   end;
end.
```

Compression de fichiers textes par conversion des tabulations.

Voir aussi

SupTab(TPCHAINE).

PtrHex

Déclaration

```
function PtrHex(P : pointer) : string;
```

Objet

Renvoie une chaîne hexadécimale comme pointeur.

Commentaires

Etant donné un pointer, *PtrHex* renvoie une chaîne de la forme 'ssss:oooo' ou 'ssss' est l'adresse du segment et 'oooo' est l'offset.

Exemple

```
if (ErrorAddr <> nil) or (ExitCode <> 0) then begin
   {Ne considère pas ^Break comme une erreur}
   if ExitCode <> $FF then
      WriteLn('Erreur ', MotHex(ExitCode),' en ', PtrHex(ErrorAddr));
   {Empêche la bibliothèque runtime d'afficher l'erreur}
   ErrorAddr := nil;
   ExitCode := 0;
end;
```

Le code ci-dessus pourrait appartenir à un gestionnaire de sorties défini par l'utilisateur pour indiquer les erreurs d'exécution. *PtHex* est utilisé pour transformer l'adresse de l'erreur en une chaîne de caractères.

RechBMSansMajMin

Déclaration

```
function RechBMSansMajMin(var Buffer ; LongBuf : Word;
                          Tb : TableB ; ChaineRech : String ) : Word;
```

Objet

Recherche de la chaîne *ChaineRech* dans le buffer sans tenir compte des majuscules ou des minuscules et en utilisant l'algorithme de Boyer-Moore.

Commentaires

Cette fonction est identique à *RechercheBM* mais la recherche ne tient compte ni des majuscules ni des minuscules. Voir *RechercheBM* pour plus de détails.

Recommandation importante : le programme appelant doit vérifier que *ChaineRech* est en majuscules, avant d'appeler *ConstruitTableBM* ou *RechBMSansMajMin.*

RechBMSansMajMin est presque toujours plus performante que *RechercheSansMajMin,* même si la table de Boyer-Moore doit être reconstruite à chaque recherche. La vitesse de *RechBMSansMajMin* peut être légèrement plus lente que celle de *RechercheSansMajMin,* mais aussi 4 fois plus rapide que celle-ci en fonction du nombre de fois où la table de Boyer-Moore doit être reconstruite de la longueur de *ChaineRech* et du contenu des données.

Notez que *RechBMSansMajMin* gère correctement tous les caractères alphabétiques dans le jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemple

```
function RBMsansMM(var Buffer; TailleBuf : Word;
                   ChnCorresp : String): Word;
var
   TableLocale : TableB;
begin
   ChnCorresp := ChnMajusc(ChnCorresp);
      {Garantit que ChnCorresp est en majuscules}
   ConstruitTableBM(ChnCorresp, TableLocale);
      {Construit la table de liens}
   RBMsansMM := RechBMSansMajMin(Buffer, TailleBuf, TableLocale,
                                 ChnCorresp);
end;
```

Cette fonction masque au programme appelant toutes les opérations préléminaires à une recherche de Boyer-Moore dépendante des majuscules et minuscules. Si la table de Boyer-Moore doit être utilisée plus d'une fois, cette fonction sera moins performante, bien qu'elle soit encore plus rapide que la plupart des autres solutions.

Voir aussi

ConstruitTableBM(TPCHAINE) RechercheBM (TPCHAINE)
RechercheSansMajMin(TPCHAINE)

Recherche

Déclaration

```
function Recherche(var Buffer; LongBuffer : Word);
                   var SousChaine; LongSousChaine : Word) : Word;
```

Objet

Recherche une sous-chaîne dans *Buffer*.

Commentaires

De portée générale, cette fonction de recherche très rapide s'applique aussi bien aux chaînes de caractères qu'aux longues séquences d'octets. La variable analysée est passée dans le paramètre *Buffer*. La longueur de la région à explorer est passée dans *LongBuffer*. La chaîne recherchée est passée dans *SousChaine*. Le nombre de caractères recherchés est passé dans *LongSousChaine*.

Si la chaîne recherchée est trouvée, *Recherche* renvoie le nombre des caractères qui ont été balayés pour atteindre le début de la sous-chaîne. Ce nombre peut être égal à zéro ou plus. Si la sous-chaîne n'est pas trouvée, la valeur $FFFF est renvoyée. Puisque les deux buffers de chaînes de caractères sont des paramètres VAR, les chaînes littérales ne peuvent être utilisées comme paramètres.

Exemple

```
var S, T : string; Posn : Word;
begin
   S := 'La démonstration commence maintenant';
   T := 'La';
   Posn := Recherche(S[1], Length(S), T[1], Length(T));
      {Posn contient 0}
   T := 'nst';
   Posn := Recherche(S[0], Length(S), T[1], Length(T));
      {Posn contient 8}
   T := 'zzz';
   Posn := Recherche(S[0], Length(S), T[1], Length(T));
      {Posn contient $FFFF}
end.
```

En démarrant la recherche à la position zéro, la fonction *Recherche* fonctionne de la même façon que la fonction *Pos* de Turbo. Néanmoins, démarrer la recherche en partant de zéro peut être dangereux puisque l'octet de longueur sera inclus dans les caractères recherchés. En indexant la chaîne *Buffer* comme le montre l'exemple, les recherches précédentes seront écartées.

Voir aussi

PosAsc (TPASCIIZ)

RechercheBM

Déclaration

```
function RechercheBM(var Buffer ; LongBuf : Word ;
                     Tb : TableB ; ChaineRech : String) : Word;
```

Objet

Recherche dans *Buffer* la chaîne *ChaineRech* avec l'algorithme de Boyer-Moore.

Commentaires

Cette routine de recherche de chaînes de caractères renvoie un résultat compatible avec la fonction *Recherche* de TPCHAINE. Néanmoins, *RechercheBM* utilise l'algorithme de Boyer-Moore qui dans certains cas conduit à une amélioration significative des performances comparées à celles de *Recherche.*

RechercheBM diffère de *Recherche* par ses paramètres. Comme *Recherche,* elle explore tout le buffer égal à 65534 octets au maximum (*LongBuf*). Contrairement à *Recherche,* elle nécessite une table spéciale (*Tb*), initialisée par *ConstruitTableBM* et la chaîne à rechercher est une chaîne Turbo normale.

RechercheBM renvoie le nombre d'octets parcourus pour retrouver la chaîne cible. Si la chaîne n'est pas trouvée, *RechercheBM* renvoie $FFFF. Si la chaîne cible est trouvée au début du buffer, *RechercheBM* renvoie 0.

Les performances de l'algorithme de recherches de Boyer-Moore, dépendent de plusieurs facteurs. Si la taille du buffer est égale à celle d'une chaîne Turbo normale (255 octets ou moins), et que la table de Boyer-Moore doit être recalculée à chaque recherche, *RechercheBM* sera toujours plus lente que la routine simple *Recherche.* D'un autre côté, si le buffer est grand, ou si la même table peut être utilisée plusieurs fois, les performances de *RechercheBM* seront soit égales à celles de *Recherche,* soit deux fois plus rapide que *Recherche,* selon la longueur de *ChaineRech.* En général, plus la chaîne à rechercher sera longue, plus la routine *RechercheBM* sera avantageuse.

Remerciements à Don Strenczewilk qui a fourni la version initiale des routines de recherches de Boyer-Moore.

Exemple

```
function RBM(var Buffer; TailleBuf : Word;
             ChnCorresp : String) : Word;
var
   TableLocale : TableB;
begin
   ConstruitTableBM(ChnCorresp, TableLocale);
   RBM := RechercheBM(Buffer, TailleBuf, TableLocale, ChnCorresp);
end;
```

Cette fonction masque la table de Boyer-Moore au programme appelant. Malheureusement, elle doit aussi mettre à jour cette table à chaque appel ce qui diminue les performances de la recherche de Boyer-Moore. Cette fonction fournira de bonnes performances seulement si le buffer est grand. Sinon, le programme devrait appeler *ConstruitTableBM* explicitement et utiliser la table calculée aussi longtemps que possible avant de la libérer.

Voir aussi

ConstruitTableBM(TPCHAINE) Recherche(TPCHAINE)
RechercheBMSansMajMin(TPCHAINE)

RechercheSansMajMin

Déclaration

```
function RechercheSansMajMin(var Buffer ; LongBuf : Word;
                             var ChnRech; LongChn : Word) : Word;
```

Objet

Recherche *ChnRech* dans Buffer sans tenir compte des majuscules et des minuscules.

Commentaires

Cette fonction est identique à *Recherche* de (TPCHAINE) mais elle ne tient compte ni des majuscules et ni des minuscules. Les caractères de *Buffer* et de *ChnRech* doivent être mis en majuscules avant la comparaison. Le passage en majuscules ne modifie pas les paramètres actuels.

Si la chaîne *ChnRech* n'est pas trouvée, *RechercheSansMajMin* renvoie $FFFF. Sinon elle renvoie le nombre de caractères balayés pour trouver le début de la chaîne.

RechercheSansMajMin s'exécute 2 à 4 fois plus lentement que *Recherche* à cause du passage en majuscules.

Voir la fonction *Recherche* de TPCHAINE pour plus d'informations et des exemples.

Notez que *RechercheSansMajMin* gère correctement tous les caractères alphabétiques du jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Voir aussi

Recherche(TPCHAINE) RechBMSansMajMin(TPCHAINE)

ReelEnChn

Déclaration

```
function ReelEnChn(R : Float; Largeur, Nb : byte) : String;
```

Objet

Convertit un réel en chaîne.

Commentaires

Cette fonction fait appel à la routine prédéfinie *Str* de *Turbo Pascal*. Elle convertit *R* en une chaîne de taille *Largeur* et contenant *Nb* décimales.

Remarque : Si TPCHAINE est compilé avec l'option $N+, cette routine acceptera des variables de type Extended, et de type Real dans le cas contraire.

Exemple

```
R := 1234.56789;
Str(R:7:2, S);
S := ReelEnChn(R, 7, 2);
```

Dans les deux cas, *S* contiendra '1234.57'.

Soundex

Déclaration

```
function Soundex(S : string) : string;
```

Objet

Renvoie 4 caractères 'soundex' de la chaîne d'entrée.

Commentaires

Le code soundex est souvent utilisé pour fournir une valeur de voisinage des noms propres. *Soundex* renvoie une chaîne de 4 caractères à partir de la chaîne d'entrée selon l'algorithme *Soundex*. Le premier caractère est le caractère majuscule du premier caractère de la chaîne d'entrée. Les trois caractères restant sont des chiffres codés à partir de la chaîne d'entrée. En comparant les deux chaînes soundex de deux noms voisins, il est possible de trouver le nom exact. *Soundex* est utile pour les chaînes composées principalement de caractères alphabétiques. Les blancs de tête doivent être enlevés avant d'appeler *Soundex*. *Soundex* convertit la chaîne d'entrée en majuscules.

Exemples

```
Soundex('macintosh');
Soundex('Mcntosh');
```

Retournent 'M253'.

SuffixeParDefaut

Déclaration

```
function SuffixeParDefaut(Nom, Suffixe: string) : string;
```

Objet

Renvoie un nom de fichier ou un nom de chemin avec un suffixe par défaut.

Commentaires

Si *Nom* comporte déjà un suffixe, aucune modification n'est opérée. Sinon *Suffixe* est ajouté à ce nom. *Suffixe* doit avoir au maximum trois caractères et ne doit pas commencer par un point.

Exemples

```
SuffixeParDefaut('TEST', 'PAS');
```

Retourne 'TEST.PAS'

```
SuffixeParDefaut('TEST.', 'PAS');
```

Retourne 'TEST.'

Voir aussi

ForceSuffixe (TPCHAINE)

SuffixeSeul

Déclaration

```
function SuffixeSeul(Nom : string) : string;
```

Objet

Renvoie simplement le suffixe d'un nom de chemin.

Commentaires

Le lecteur, le répertoire et le nom de fichier sont enlevés du nom de chemin et le suffixe seul est renvoyé. Le suffixe n'inclut pas de point. Si la chaîne d'entrée ne comporte pas de suffixe, la chaîne vide est renvoyée.

Exemples

```
SuffixeSeul('TEST.PAS');
```

Retourne 'PAS'

```
SuffixeSeul('C:\FDON');
```

Retourne une chaîne vide ''.

Voir aussi

NomFichSeul (TPCHAINE) NomCheminSeul (TPCHAINE)

SupTab

Déclaration

```
function SupTab (S : String ; TailleTab : Byte) : string;
```

Objet

Supprime les tabulations de la chaîne *S* et les remplace par des blancs sur une longueur de *TailleTab*.

Commentaires

SupTab supprime les tabulations d'une chaîne et les convertit en une suite de blancs.

Contrairement à *PlaceTab*, *SupTab* convertit toute la chaîne, qu'elle contienne ou non des littéraux entre apostrophes.

TailleTab = 8 est la taille normale utilisée par la plupart des éditeurs de textes. Si *TailleTab* vaut 0, *SubTab* renverra toujours une chaîne vide.

Exemple

```
var
   S : string;
begin
   while not EOF(Input) do begin
      ReadLn(Input, S);
      WriteLn(Output, SupTab(S, 8));
   end;
end.
```

Filtre de conversion des tabulations dans un fichier texte.

Voir aussi

PlaceTab(TPCHAINE)

UpCase

Déclaration

```
function UpCase(Car : Char) : Char;
```

Objet

Renvoie *Car* en majuscules.

Commentaires

Cette routine convertit les caractères en majuscules. Contrairement à la fonction standard *UpCase* de l'unité SYSTEM et à la fonction *MacroMajuscule, UpCase* de TPCHAINE gère correctement tous les caractères alphabétiques du jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemple

```
UpCase('a');
```

Renvoie 'A'

```
UpCase(#164); {'n' et '~'}
```

Renvoie #165('N' avec un tilde).

Voir aussi

Minuscule(TPCHAINE) ChnMajusc(TPCHAINE)

TPASCIIZ : Chaînes longues ASCIIZ

L'unité TPASCIIZ définit un nouveau type de chaînes pour Turbo Pascal. Contrairement aux chaînes normales de Turbo qui sont limitées à 255 octets, une chaîne de type ASCIIZ peut contenir jusqu'à 65520 caractères.

Une chaîne ASCIIZ est définie comme celles qui sont utilisées par MS-DOS pour les opérations internes et par le langage de programmation 'C'. Une chaîne ASCIIZ est simplement un tableau de caractères terminé par un caractère nul (ASCII #0).

Deux inconvénients peuvent arriver en utilisant des chaînes ASCIIZ. Le premier est que la longueur d'une telle chaîne ne peut être définie immédiatement. Une routine doit balayer toute la chaîne jusqu'à ce qu'elle rencontre un caractère nul. TPASCIIZ implémente cette routine de recherche avec une macro inline Turbo et utilise l'instruction SCASB du 8088 pour obtenir la meilleur performance possible pour cette opération. Il en résulte que la performance demandée à l'opération de recherche de longueur ne devrait pas constituer un problème en pratique.

Le second inconvénient peut s'appliquer à l'utilisation des chaînes de caractères pour certaines applications. Comme le caractère nul termine une chaîne ASCIIZ, les caractères nuls ne peuvent faire partie de la chaîne elle-même. Ainsi, les chaînes ASCIIZ ne peuvent pas contenir de séquences de codes de balayage étendus (qui commencent par un caractère nul) sauf si les caractères nuls sont préalablement convertis en un autre caractère.

Bien que les chaînes ASCIIZ puissent contenir jusqu'à 65520 caractères, l'unité TPASCIIZ définit une longueur maximum par défaut de 1024 caractères. Cette limite vous permettra de déclarer le nombre de chaînes ASCIIZ ou variables globales sans utiliser le segment de 64K tout entier. Si vous avez besoin de chaînes plus longues, modifiez la constante *MaxAsciiz*. Dans ce cas, les chaînes ASCIIZ devront probablement être allouées sur le tas.

Constantes

```
MaxAsciiz = 1024;
```

Longueur maximum d'une chaîne ASCIIZ. Cette valeur peut atteindre 65520 caractères.

```
NonTrouve = $FFFF;
```

Valeur retournée par *PosAsc* et *PosChn* si la sous-chaîne cible n'est pas trouvée.

Types

```
Asciiz = array[0..MaxAsciiz] of Char;
```

La représentation interne d'une chaîne ASCIIZ. Une position supplémentaire est utilisée pour enregistrer l'octet de terminaison nul.

```
AsciizPtr = ^Asciiz;
```

Un pointeur sur une chaîne ASCIIZ. Ces chaînes étant souvent assez longues, il est préférable de les allouer sur le tas.

AscCaracChn

Déclaration

```
procedure AscCaracChn(Car : Char; Longueur : Word; var A : Asciiz);
```

Objet

Renvoie une chaîne ASCIIZ de longueur *Longueur*, remplie avec *Car*.

Commentaires

Si *Longueur* est égale à 0, une chaîne nulle est renvoyée. Sinon une chaîne ASCIIZ de longueur *Longueur* est renvoyée, remplie avec le caractère *Car*.

Exemple

```
AscCaracChn('-', 10, A);
```

A contient '----------'.

Voir aussi

ChaineCh (TPCHAINE)

AscCentre

Déclaration

```
procedure AscCentre(var A : Asciiz; Largeur : Word; var B : Asciiz);
```

Objet

Renvoie une chaîne ASCIIZ centrée dans une chaîne ASCIIZ de blancs avec une largeur spécifiée.

Commentaires

Si *A* excède déjà *Largeur*, le paramètre *B* retourné est le même que *A*. Sinon *B* renvoie une chaîne ASCIIZ d'une largeur spécifiée avec *A* centrée dans un champ de blancs.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   AscCentre(A, 10, B); {B prend la valeur '  Bonjour '}
end.
```

Voir aussi

AscCentreCar (TPASCIIZ) Centre (TPCHAINE)

AscCentreCar

Déclaration

```
procedure AscCentreCar(Var A:Asciiz; Ch:Char; Largeur:Word; var
B:Asciiz);
```

Objet

Renvoie une chaîne ASCIIZ centrée dans une chaîne ASCIIZ de *Ch* sur une largeur spécifiée.

Commentaires

Si *A* dépasse déjà *Largeur*, la chaîne *B* retournée est la même que *A*. Sinon, *B* est renvoyée comme une chaîne ASCIIZ de largeur spécifiée ; *A* centré dans un champ de *Ch*.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   AscCentreCar(A, '-', 10, B); {B prend la valeur '--Bonjour-'}
end.
```

Voir aussi

AscCentre (TPASCIIZ) CentreCar (TPCHAINE)

AscComplete

Déclaration

```
procedure AscComplete(var A : Asciiz; Longueur : Word;
                      var B : Asciiz);
```

Objet

Complète à droite la chaîne *A* sur une longueur *Longueur* avec des blancs et renvoie le résultat dans *B*.

Commentaires

Si la longueur de *A* est supérieure ou égale à *Longueur*, alors *B* prend la valeur de *A*. Sinon, *B* prend la valeur de *A* et des blancs sont ajoutés à la fin de *B* jusqu'à ce que la longueur *Longueur* soit atteinte.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   AscComplete(A, 10, B);  {B prend la valeur 'Bonjour   '}
end.
```

Voir aussi

AscCompleteGauche (TPASCIIZ)
AscCompleteCar (TPASCIIZ)
CompleteCar (TPCHAINE)

AscCompleteCar

Déclaration

```
procedure AscCompleteCar(var A : Asciiz; Car : Char; Longueur : Word;
                         var B : Asciiz);
```

Objet

Complète à droite la chaîne *A* sur une longueur *Longueur* avec *Car* et renvoie le résultat dans *B*.

Commentaires

Si la longueur de *A* est supérieure ou égale à *Longueur*, alors *B* prend la valeur de *A*. Sinon, *B* prend la valeur de *A* et les caractères *Car* sont ajoutés à la fin de *B* jusqu'à ce que la longueur *Longueur* soit atteinte.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   AscCompleteCar(A, '-', 10, B); {B prend la valeur 'Bonjour---'}
end.
```

Voir aussi

AscCompleteGaucheCar (TPASCCIZ) CompleteCar (TPCHAINE)
AscComplete (TPASCIIZ)

AscCompleteGauche

Déclaration

```
procedure AscCompleteGauche(var A : Asciiz; Longueur : Word;
                            var B : Asciiz);
```

Objet

Complète à gauche la chaîne ASCIIZ *A* sur une longueur *Longueur* avec des blancs et renvoie le résultat dans *B*.

Commentaires

Si la longueur de *A* supérieure ou égale *Longueur*, alors *B* renverra la chaîne *A* elle-même. Sinon *A* est transférée dans *B* et les caractères *Ch* sont ajoutés au début de *B* jusqu'à ce que la longueur *Longueur* soit atteinte.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour',A);
   AscCompleteGauche(A, 10, B); {B prend la valeur '   Bonjour'}
end.
```

Voir aussi

AscComplete (TPASCIIZ) CompleteGauche (TPCHAINE)
AscCompleteGaucheCar (TPASCIIZ)

AscCompleteGaucheCar

Déclaration

```
procedure AscCompleteGaucheCar(var A : Asciiz; Car : Char;
                               Longueur : Word; var B : Asciiz);
```

Objet

Complète à gauche la chaîne *A* sur une longueur *Longueur* avec *Car* et renvoie le résultat dans *B*.

Commentaires

Si la longueur de *A* est supérieure ou égale à *Longueur*, alors *B* contiendra la chaîne *A*. Sinon, *A* est transférée dans *B* et les caractères *Car* sont ajoutés au début de *B* jusqu'à ce que la longueur *Longueur* soit atteinte.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   AscCompleteGaucheCar(A, '-', 10, B); {B prend la valeur
                        '---Bonjour'}
end.
```

Voir aussi

AscCompleteGauche (TPASCIIZ) CompleteGaucheCar (TPCHAINE)
AscCompleteCar (TPASCIIZ)

AscDepuisTas

Déclaration

```
procedure AscDepuisTas(P : AsciizPtr; var A : Asciiz);
```

Objet

Renvoie une chaîne ASCII à partir du tas et une chaîne vide si le pointeur est à nil.

Commentaires

Puisque les chaînes ASCIIZ peuvent être très longues, il est souvent conseillé de les enregistrer sur le tas plutôt que dans le segment de données où elles peuvent rapidement occuper les 64 K octets disponibles. *AscDepuisTas* fonctionne conjointement avec *AscSurTas* et *DeallouéAsc*.

AscSurTas alloue de la place sur le tas pour une chaîne ASCII et renvoie un pointeur sur cette chaîne. *AscDepuisTas* renvoie les données à partir du tas dans une autre variable. *DeallouéAsc* libère la place sur le tas lorsqu'elle n'est plus nécessaire.

Si le pointeur *P* est à nil, *AscDepuisTas* renvoie une chaîne vide.

Remarquez que l'instruction

```
A := P^;
```

Donne le même résultat que

```
AscDepuisTas(P, A);
```

Quoiqu'il en soit, l'appel à *AscDepuisTas* peut être plus rapide puisqu'elle déplace seulement la donnée actuelle plutôt que le nombre total d'octets dans la structure.

Exemple

```
var
   P : AsciizPtr;
   A : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   P := AscSurTas(A);
   AscDepuisTas(P, A);
end.
```

Voir aussi

AscSurTas (TPASCIIZ)
DeallouéAsc (TPASCIIZ)
ChaineDuTas (TPCHAINE)

AscEff

Déclaration

```
procedure AscEff(var A, B : Asciiz);
```

Objet

Renvoie une chaîne ASCII en éliminant les espaces de tête et de fin.

Commentaires

Le caractère espace est considéré ici comme n'importe quel caractère ASCII dont le rang est inférieur ou égal à 32 sauf le caractère #0 qui est utilisé comme caractère de fin des chaînes ASCIIZ. Si la chaîne *A* ne contient que des espaces, une chaîne nulle est retournée dans *B*. Les espaces internes à *A* ne sont pas modifiés.

Exemple

```
var
   A,B : Asciiz;
begin
   ChnEnAsc('  Bonjour à tous !  ', A);
   AscEff(A, B); {B prend la valeur 'Bonjour à tous !'}
end.
```

Voir aussi

AscEffDeb (TPASCIIZ)
AscEffFin (TPASCIIZ)
EffBlancs (TPCHAINE)

AscEffDebut

Déclaration

```
procedure AscEffDebut(var A, B : Asciiz);
```

Objet

Renvoie une chaîne ASCII dont les espaces de tête ont été supprimés.

Commentaires

Le caractère espace est considéré ici comme n'importe quel caractère ASCII dont le rang est inférieur ou égal à 32 sauf le caractère #0 qui est utilisé comme caractère de fin des chaînes ACIIZ. Si la chaîne *A* ne contient que des espaces, une chaîne nulle est retournée dans *B*. Des espaces internes à la chaîne *A* ne sont pas modifiés.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('  Bonjour à tous !  ', A);
   AscEffDebut(A, B); {B prend la valeur 'Bonjour à tous !  '}
end.
```

Voir aussi

AscEff (TPASCIIZ)
AscEffFin (TPASCIIZ)
EffEspDeb (TPCHAINE)

AscEffFin

Déclaration

```
procedure AscEffFin(var A, B : Asciiz);
```

Objet

Renvoie une chaîne ASCIIZ dont les espaces de fin ont été enlevés.

Commentaires

Le caractère espace est défini ici comme n'importe quel caractère ASCII dont le rang est inférieur ou égal à 32 sauf le caractère #0 qui est utilisé comme caractère de fin des chaînes AZSCIIZ. Si la chaîne *A* est composée uniquement d'espaces, une chaîne nulle est retournée dans *B*. Les espaces internes à *A* ne sont pas modifiés.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('  Bonjour à tous !  ', A);
   AscEffFin(A, B); {B prend la valeur '  Bonjour à tous !'}
end.
```

Voir aussi

AscEff (TPASCIIZ)
AscEffDeb (TPASCIIZ)
EffEspFin (TPCHAINE)

AscEnChn

Déclaration

```
function AscEnChn(var A : Asciiz) : string;
```

Objet

Convertit une chaîne ASCIIZ en une chaîne Turbo.

Commentaires

Si la chaîne ASCIIZ dépasse 255 caractères, elle est tronquée avant de l'affecter à la chaîne Turbo.

Exemple

```
var
   A : Asciiz;
   S : string;
begin
   ChnEnAsc('Bonjour', A);
   S := AscEnChn(A); {S prend la valeur 'Bonjour'}
end.
```

Voir aussi

ChnEnAsc (TPASCIIZ)

AscMajusc

Déclaration

```
procedure AscMajusc(var A, B : Asciiz);
```

Objet

La chaîne *A* est mise en majuscules puis retournée dans *B*.

Commentaires

Chaque caractère de *A* est converti en majuscule et le résultat transmis dans *B*. Notez que *AscMajusc* gère correctement tous les caractères alphabétiques du jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour !', A);
   AscMajusc(A, B); {B prend la valeur 'BONJOUR !'}
end.
```

Voir aussi

AscMinusc (TPASCIIZ) ChnMajusc (TPCHAINE)

AscMinusc

Déclaration

```
procedure AscMinusc(var A, B : Asciiz);
```

Objet

Convertit la chaîne *A* en minuscules et renvoie le résultat dans *B*.

Commentaires

Chaque caractère de A est mis en minuscules et le résultat renvoyé dans *B*. Remarquez que *AscMinusc* gère correctement tous les caractères alphabétiques du jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('BONJOUR', A);
   AscMinusc(A, B); {B prend la valeur 'bonjour'}
end.
```

Voir aussi

ChnMinusc (TPCHAINE)

AscSurTas

Déclaration

```
function AscSurTas(var A : Asciiz) : AsciizPtr;
```

Objet

Place la chaîne ASCIIZ sur le tas et renvoie un pointeur égal à nil s'il n'y a pas assez de place.

Commentaires

Comme les chaînes ASCIIZ peuvent être très longues, il est souvent préférable de les enregistrer sur le tas plutôt que dans le segment de données où elles peuvent rapidement occuper la totalité des 64 K octets disponibles. *AscSurTas* fonctionne conjointement avec *AscDepuisTas* et *DeallouéAsc*.

AscSurTas alloue de la place sur le tas pour les chaînes ASCIIZ et renvoie un pointeur sur cette chaîne. *AscDepuisTas* renvoie la donnée à partir du tas dans une autre variable. *DeallouéAsc* libère la place occupée sur le tas lorsqu'elle n'est plus nécessaire.

S'il n'y a pas assez de place sur le tas, *AscSurTas* renvoie un pointeur à nil. Remarquez que *AscSurTas* alloue exactement la place nécessaire pour la chaîne ASCIIZ passée en paramètre. Si vous voulez allouer plus de place pour des chaînes ASCIIZ plus longues, utilisez l'instruction suivante :

```
New(P); {Où P est de type AsciizPtr}
```

Exemple

```
var
   P : AsciizPtr;
   A : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   P := AscSurTas(A); {P pointe sur la valeur 'Bonjour'#0}
end.
```

Voir aussi

AscDepuisTas (TPASCIIZ)
ChaineDuTas (TPCHAINE)
DeallouéAsc (TPASCIIZ)

ChnEnAsc

Déclaration

```
procedure ChnEnAsc(S : string; var A : Asciiz);
```

Objet

Convertit une chaîne Turbo en une chaîne ASCIIZ.

Commentaires

Cette procédure fournit un moyen pratique pour initialiser des chaînes ASCIIZ. Elle convertit une chaîne Turbo normale (au plus 255 caractères) en une chaîne ASCIIZ. La procédure peut être également pratique pour interfacer les procédures Turbo aux différentes fonctions MS-DOS qui appellent des chaînes ASCIIZ. N'oubliez pas que tout caractère nul (ASCII #0) inclus dans une chaîne Turbo entraînera une terminaison prématurée d'une chaîne ASCIIZ.

Exemple

```
var
   A : Asciiz;
begin
   ChnEnAsc('Bonjour', A); {A prend la valeur 'Bonjour'}
end.
```

Voir aussi

AscEnChn (TPASCIIZ)

CompareAsc

Déclaration

```
function CompareAsc(var A1, A2 : Asciiz) : TypeCompareAsc;
```

Objet

Renvoie *Inferieur, Egal* ou *Superieur* si A1<A2, A1=A2, ou A1>A2.

Commentaires

Deux chaînes ASCIIZ sont comparées en utilisant les mêmes règles que pour les chaînes normales. La fonction renvoie un type énuméré dont les valeurs sont *AscInferieur, AscEgal* et *AscSuperieur*.

Exemple

Voir *CompareChaine* pour des exemples (TPCHAINE).

Voir aussi

CompareChaine (TPCHAINE) CompareAscMajusc (TPASCIIZ)

CompareAscMajusc

Déclaration

```
function CompareAscMajusc(var A1, A2 : Asciiz) : TypeCompareAsc;
```

Objet

Compare deux chaînes ASCIIZ converties en majuscules.

Commentaires

Deux chaînes ASCIIZ sont comparées en utilisant les mêmes règles que pour les chaînes normales. Pendant la comparaison, tous les caractères minuscules sont convertis en majuscules sans modifier les paramètres actuels. La fonction renvoie un type énuméré dont les valeurs sont *AscInferieur, AscEgal* et *AscSuperieur*. Notez que *CompareAscMajusc* gère correctement tous les caractères alphabétiques du jeu de caractères international ('A'..'Z', 'a'..'z', #128..#154, #160..#167).

Exemple

Voir *CompareChnMajusc* pour des exemples (TPCHAINE).

Voir aussi

CompareAsc (TPASCIIZ)

CompareChnMajusc (TPCHAINE)

ConcatAsc

Déclaration

```
procedure ConcatAsc(var A, B, C : Asciiz);
```

Objet

Concatène deux chaînes de caractères ASCIIZ.

Commentaires

Cette procédure est semblable à l'expression de chaîne normale C := A+B. Si la chaîne résultante dépasse la taille maximum des chaînes ASCIIZ, le résultat est tronqué.

Exemple

```
var
   A, B, C : Asciiz;
begin
   ChnEnAsc('Bonjour ', A);
   ChnEnAsc('à tous !', B);
   ConcatAsc(A, B, C);  {C prend la valeur 'Bonjour à tous !'}
end.
```

Voir aussi

ConcatChn (TPASCIIZ)

ConcatChn

Déclaration

```
procedure ConcatChn(var A : Asciiz, S : string; var C : Asciiz);
```

Objet

Concatène une chaîne à une chaîne ASCIIZ.

Commentaires

Cette procédure est semblable à l'expression normale de chaîne C := A+S. Dans ce cas, une chaîne normale Turbo est concaténée à une chaîne ASCIIZ A et le résultat renvoyé dans la chaîne ASCIIZ C. Si le résultat ASCIIZ dépasse la longueur maximum, cette chaîne est tronquée.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour ', A);
   ConcatChn(A, 'à tous !', B);  {B prend la valeur
            'Bonjour à tous !'}
end.
```

Voir aussi

ConcatAsc (TPASCIIZ)

CopieAsc

Déclaration

```
procedure CopieAsc(var A : Asciiz; Debut, Longueur : Word; var O :
Asciiz);
```

Objet

Renvoie une sous-chaîne appartenant à *A*.

Commentaires

Cette procédure est semblable à la fonction *Copy* sur les chaînes normales. *Longueur* caractères sont extraits à partir de la position *Debut* dans la chaîne A et renvoyés dans O. Remarquez que *Debut = 0* référencera le premier caractère de *A*.

S'il y a plus de caractères demandés que *A* n'en contient, les caractères existants sont renvoyés. Si la position *Debut* est supérieure à la longueur de A, la chaîne nulle est renvoyée.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   CopieAsc(A, 1, 3, B); {B prend la valeur 'onj'}
   CopieAsc(A, 6, 10, B); {B prend la valeur 'r'}
end.
```

DeallouеAsc

Déclaration

```
procedure DeallouеAsc(var P : AsciizPtr);
```

Objet

Libère un emplacement sur le tas pointé par *P*.

Commentaires

Comme les chaînes ASCIIZ peuvent être longues, il est souvent préférable de les enregistrer sur le tas plutôt que dans un segment de données où elles peuvent rapidement occuper les 64 K octets disponibles. *DeallouеAsc* fonctionne conjointement avec *AscSurTas*.

AscSurTas alloue de l'espace sur le tas pour une chaîne ASCIIZ et renvoie un pointeur sur cette chaîne. *DeallouеAsc* libère l'espace sur le tas lorsque cet emplacement n'est plus nécessaire.

Si le pointeur *P* est à nil, *DeallouеAsc* ne fait rien. *DeallouеAsc* ne devrait être utilisé que pour libérer des pointeurs alloués directement par une instruction *New*. Dans la plupart des cas, utilisez la procédure *Dispose* interne de Turbo pour libérer de l'espace sur le tas.

Exemple

```
var
   P : AsciizPtr;
   A : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   P := AscSurTas(A);
   DeallouеAsc(P);  {8 octets sont libérés et P prend la valeur Nil}
end.
```

Voir aussi

AscSurTas (TPASCIIZ) DeallouеChaine (TPCHAINE)

EcritAsc

Déclaration

```
function EcritAsc(var F : Text; var A : Asciiz) : Boolean;
```

Objet

Ecrit une chaîne ASCIIZ sur un fichier texte et renvoie "True" en cas de succès.

Commentaires

Cette routine vous permet d'écrire des longues chaînes de caractères sur des fichiers textes. *F* est n'importe quel fichier texte Turbo précédemment ouvert en écriture. La chaîne *A* ASCIIZ est écrite sur le fichier sans le caractère nul de terminaison. Si une erreur d'entrée/sortie se produit, la fonction renverra "False".

Exemple

```
var
   F : text;
   A : Asciiz;
begin
   Assign(F, 'LONG.TXT');
   Reset(F);                                  {Ouverture en lecture}
   while not Eof(F) do begin                  {Lecture tant que pas fin du
                                               fichier}
      if not LitAsc(F, A) then                {Lecture ligne suivante}
         Halt;                                {Arrêt si erreur}
      if not EcritAsc(OutPut, A) then         {Ecriture de A sur l'écran}
         Halt;                                {Arrêt si erreur}
      writeln;                                {Ajout du retour chariot}
   end;
end.
```

Voir aussi

LitAsc (TPASCIIZ)

InsereAsc

Déclaration

```
procedure InsereAsc(var Obj, A : Asciiz; Debut : Word);
```

Objet

Insère la chaîne *Obj* ASCIIZ à la position *Debut* dans *A*.

Commentaires

Cette routine est semblable à la procédure *Insert* pour les chaînes normales. La chaîne Obj est insérée à la position *Debut* dans la chaîne ASCIIZ *A*. Remarquez que *Debut = 0* référence la première position de *A*.

Si l'insertion d'*Obj* entraîne un dépassement de la longueur maximum de *A* pour les chaînes ASCIIZ, la chaîne *A* est tronquée. Si *Debut* dépasse la longueur de *A*, la chaîne *Obj* est concaténée à *A*.

Exemple

```
var
   A, B : Asciiz;
begin
   ChnEnAsc('Bonjour ', A);
   ChnEnAsc('à tous !', B);
   InsereAsc(A, B, 0); {B prend la valeur 'Bonjour à tous !'}
end.
```

Voir aussi

InsereChn (TPASCIIZ)

InsereChn

Déclaration

```
procedure InsereChn(Obj : string; var A : Asciiz; Debut : Word);
```

Objet

Insère la chaîne *Obj* à la position *Debut* dans *A*.

Commentaires

Cette routine est semblable à la procédure *Insert* pour les chaînes normales. La chaîne normale *Obj* est insérée à la position *Debut* dans la chaîne ASCIIZ *A*. Remarquez que *Debut* = *0* référence la première position dans *A*.

Si l'insertion d'*Obj* entraîne un dépassement de la longueur maximum des chaînes ASCIIZ pour *A*, *A* est tronqué. Si *Debut* dépasse la longueur de *A*, la chaîne *Obj* est concaténée à *A*.

Exemple

```
var
   A : Asciiz;
begin
   ChnEnAsc('à tous !', A);
   InsereChn('Bonjour ', A, 0); {A prend la valeur 'Bonjour à tous'}
end.
```

Voir aussi

InsereAsc (TPASCIIZ)

LitAsc

Déclaration

```
function LitAsc(var F : Text; var A : Asciiz) : Boolean;
```

Objet

Lit une chaîne ASCIIZ sur un fichier texte et renvoie "True" en cas de succès.

Commentaires

Cette fonction vous permet de lire des chaînes longues sur des fichiers textes. *F* est un fichier texte précédemment ouvert en lecture avec n'importe quelle taille de buffer. Le paramètre *A* renverra la chaîne ASCIIZ à partir du fichier se terminant à la prochaine combinaison retour chariot/fin de ligne (ASCII #13#10).

Si une erreur d'entrée/sortie se produit, *LitAsc* renverra "False". Si *F* est déjà en fin de fichier, *LitAsc* renvoie "False" et *A* est la chaîne nulle. Si la ligne de texte dépasse la longueur maximum pour une chaîne ASCIIZ, le reste de cette ligne sera tronquée.

Exemple

```
var
   F : text;
   A : Asciiz;
begin
   Assign(F, 'LONG.TXT');
   Reset(F);                                {Ouverture en lecture}
   while not Eof(F) do begin                {Lecture tant que pas fin du
                                            fichier}
      if not LitAsc(F, A) then              {Lecture ligne suivante}
         Halt;                              {Arrêt si erreur}
      if not EcritAsc(OutPut, A) then       {Ecriture de A sur l'écran}
         Halt;                              {Arrêt si erreur}
      writeln;                              {Ajout du retour chariot}
   end;
end.
```

Voir aussi

EcritAsc (TPASCIIZ)

LongAsc

Déclaration

```
function LongAsc(A : Asciiz) : Word;
```

Objet

Renvoie la longueur d'une chaîne ASCIIZ.

Commentaires

La longueur d'une chaîne ASCIIZ est le nombre de ses caractères non compris le caractère nul qui délimite la chaîne. La longueur est donc comprise entre 0 et *MaxAsciiz*.

Comme *LongAsc* exécute une opération de comptage, elle est implémentée dans une macro inline en assembleur. De ce fait, elle occupe 12 à 14 octets de plus qu'un appel à une fonction normale. De plus, comme chaque recherche de la fin d'une chaîne prend du temps, la fonction *LongAsc* devrait être appelée un minimum de fois et en affectant son résultat à une variable de type Word.

Exemple

```
var
   A : Asciiz;
   L : Word;
begin
   ChnEnAsc('Bonjour', A);
   L := LongAsc(A); {L prend la valeur 7}
   SupprimeAsc(A, 0, 7);
   L := LongAsc(A); {L prend la valeur 0}
end.
```

PosAsc

Déclaration

```
function PosAsc(var Obja, A : Asciiz) : Word;
```

Objet

Renvoie la position de *Obja* dans *A*.

Commentaires

La routine *PosAsc* est semblable à la fonction *Pos* pour les chaînes Turbo normales. Elle renvoie la position de *Obja* dans *A*. Remarquez que la première position dans *A* est 0. Si *Obja* n'est pas trouvé dans *A*, alors *PosAsc* renvoie $FFFF (constante *NonTrouve*).

Exemple

```
var
   A, B : Asciiz;
   Posn : Word;
begin
   ChnEnAsc('Bonjour à tous !', A);
   ChnEnAsc('bichtruc', B);
   Posn := PosAsc(B, A);    {Posn prend la valeur $FFFF}
   ChnEnAsc('Bonjour', B);
   Posn := PosAsc(B, A);    {Posn prend la valeur 0}
end.
```

Voir aussi

PosChn (TPASCIIZ)

Recherche (TPCHAINE)

PosChn

Déclaration

```
function PosChn(Obj : string; var A : Asciiz) : Word;
```

Objet

Renvoie la position de la chaîne *Obj* dans *A* et renvoie la constante *NonTrouve* en cas d'échec.

Commentaires

La routine *PosChn* est semblable à la fonction *Pos* pour les chaînes Turbo normales. Elle renvoie la position de la chaîne Turbo normale *Obj* dans la chaîne ASCIIZ *A*. Remarquez que la première position de *A* est 0. Si *Obj* n'appartient pas *A*, la fonction *PosChn* renvoie $FFFF (constante *NonTrouve*).

Exemple

```
var
   A : Asciiz;
   Posn : Word;
begin
   ChnEnAsc('Bonjour à tous !', A);
   Posn := PosChn('bichtruc', A); {Posn prend la valeur $FFFF}
   Posn := PosChn('Bonjour', A);  {Posn prend la valeur 0}
end.
```

Voir aussi

PosAsc (TPASCIIZ) Recherche (TPCHAINE)

SupprimeAsc

Déclaration

```
procedure SupprimeAsc(var A : Asciiz; Debut, Longueur : Word);
```

Objet

Supprime *Longueur* caractères de *A* à partir de la position *Debut*.

Commentaires

Cette procédure est semblable à la fonction *Delete* sur les chaînes normales. *Longueur* caractères sont supprimés à partir de la position *Debut* de *A*. Remarquez que *Debut = 0* référence le premier caractère de *A*.

S'il y a plus de caractères à supprimer qu'il n'y en a dans *A*, tous les caractères existants sont supprimés. Si la position *Debut* dépasse la longueur de *A*, alors *A* n'est pas modifiée.

Exemple

```
var
   A : Asciiz;
begin
   ChnEnAsc('Bonjour', A);
   SupprimeAsc(A, 10, 10);  {A est inchangé}
   SupprimeAsc(A, 1, 1);    {A prend la valeur 'Bnjour'}
   SupprimeAsc(A, 0, 2);    {A prend la valeur 'jour'}
end.
```

TPCHNFOR : Chaînes formatées

Comme nous l'avons vu, TPCHNFOR est une très petite unité. Elle renvoie une seule variable globale (TpChn) et comporte deux routines (*LitChn* et *RetourneChn*) mais sa simplicité est toutefois un peu trompeuse. TPCHNFOR profite de la puissance des nouvelles caractéristiques de Turbo 4.0 : "Les modules d'entrée/sortie sur fichier texte" permettent d'utiliser toutes les possibilités de formatage de l'instruction *Write* sans envoyer les sorties sur un fichier ou un périphérique.

L'utilisation de TPCHNFOR est très simple : tout ce que vous avez à faire est de l'ajouter dans la clause USES au début de votre programme. L'unité TPCHNFOR s'initialisera elle-même automatiquement au début du programme. Pour créer des chaînes formatées, utilisez une instruction *Write* normale. Par exemple :

```
Write(TpChn, 'TempsEcoulé: ', heures:2, ' , Minutes:2,  ' minutes.');
```

Vous pouvez alors retrouver la chaîne formatée dè deux façons. Vous pouvez appeler *LitChn* pour placer la chaîne dans une variable ou vous pouvez appeler *RetourneChn* :

```
LitChn(MaChaine);
MaChaine := RetourneChn;
```

LitChn est plus efficace que *RetourneChn* (en termes de vitesse et d'utilisation de la pile) lorsque vous devez simplement placer la chaîne dans une variable. *RetourneChn* est plus souple puisque vous pouvez utiliser la chaîne qu'elle renvoie comme un argument dans un appel de procédure ou de fonction.

Variables

```
TpChn : Text;
```

C'est le périphérique spécifique sur lequel vous écrivez pour créer des chaînes formatées en utilisant TPCHNFOR.

Par exemple,

```
Write(TpChn, Pi:0:2);
```

créera une chaîne contenant '3.14'.

LitChn

Déclaration

```
procedure LitChn(var S : string);
```

Objet

'Lit' une chaîne dans *S* à partir d'un autre buffer.

Commentaires

LitChn charge le contenu actuel du buffer associé à *TpChn* dans une variable chaîne et vide le buffer. Au cas où le résultat n'est pas nécessaire immédiatement, elle est plus rapide et plus efficace que *RetourneChn*.

Exemple

```
Reel1 := 6.95;
Write(TpChn, 'Taux horaire ',Reel1:0:2);
LitChn(MaChaine);
```

Chaine contiendra 'Taux : 6.95F par heure' après l'appel à *LitChn*.

Voir aussi

RetourneChn (TPCHNFOR)

RetourneChn

Déclaration

```
function RetourneChn : string;
```

Objet

Renvoie le contenu de notre buffer de chaîne.

Commentaires

RetourneChn renvoie le contenu actuel du buffer associé à *TpChn* et vide le buffer. Contrairement à *LitChn*, elle peut être utilisée pour passer le contenu du buffer de chaîne à une autre procédure ou fonction sans utiliser une variable intermédiaire.

Exemple

```
Reel1 := 6.95;
Write(TpChn, 'Taux horaire ',Reel1:0:2);
EcritVite(RetourneChn, 1, 1, TextAttr);
```

L'argument chaîne passé à *EcritVite* sera 'Taux : 6.95F par heure'.

Voir aussi

LitChn (TPCHNFOR)

5 Appels au DOS et au BIOS

Généralités

Même si *Turbo Pascal* a permis depuis longtemps aux programmeurs d'accéder facilement au DOS et au BIOS, dans certains cas, vous avez dû écrire vos propres routines. L'unité DOS standard de *Turbo Pascal* 4.0 et 5.0 a considérablement allégé la charge des programmeurs mais laisse néanmoins un grand nombre de cas non résolus. L'unité TPDOS de *Turbo Professionnel* comble la plupart de ces lacunes.

Contrairement à l'unité TPCRT qui remplace l'unité CRT de Turbo, TPDOS complète simplement l'unité DOS. Ces deux unités contiennent la plupart des routines nécessaires pour exploiter la puissance du DOS et du BIOS.

La première fonction de TPDOS est de gérer les fichiers et les accès disques. Entre autre chose, vous trouverez des routines pour : compter le nombre de lecteurs installés, sélectionner un nouveau lecteur, lire et écrire des secteurs de disques absolus, lire et modifier l'adresse de transfert disque courante (DTA), activer/désactiver la vérification d'écriture sur disque et tester l'existence d'un fichier, soit dans le répertoire courant, soit n'importe où sur le PATH du DOS. Vous trouverez aussi des routines qui vous permettront de lire aléatoirement des fichiers textes : TPDOS contient des routines équivalentes aux routines *Seek, FileSize, FilePos* et *Flush* de *Turbo Pascal.*

TPDOS contient également une variété de routines qui vous permettent d'accéder aux possibilités d'impression concurrente du DOS 3.x. Lorsque vous avez vérifié que PRINT.COM est installé en appelant la fonction *PrintPresent,* vous pouvez envoyer un fichier dans le spooler d'impression simplement en appelant *LanceImpFich* et vous pouvez arrêter l'impression aussi facilement en appelant *AnnuleImpFich* ou *AnnuleImpTousFichs.* Vous pouvez même examiner la liste des fichiers en attente d'impression en appelant *LitStatutImp.*

La routine la plus originale de TPDOS est *ExecDos.* Bien que la routine *Exec* de l'unité DOS de Turbo soit déjà satisfaisante, elle pose deux problèmes. Tout d'abord elle suppose, dans de nombreux cas, que vous avez localisé COMMAND.COM en recherchant dans l'environnement DOS le

paramétrage de COMSPEC ; si vous ne savez pas comment trouver COMMAND.COM vous ne pouvez pas "accéder au DOS" en utilisant Exec. Deuxièmement, elle vous oblige à définir à la compilation l'espace maximum nécessaire sur le tas, pour votre programme.

ExecDos localise automatiquement COMMAND.COM. Tout ce que vous devez faire est de lui passer le nom de la commande DOS à exécuter et un indicateur booléen précisant si vous avez donné ou non le nom de chemin complet d'un fichier exécutable. Plus important, *ExecDos* vous permet d'allouer toute la mémoire disponible sans restrictions. Avant d'exécuter votre commande DOS, *ExecDos* place la table des fragments Turbo dans la mémoire basse et renvoie tout l'espace situé au-dessus du sommet courant du tas réservé au DOS. Lorsque la commande a été exécutée, toute la mémoire allouée au DOS est redistribuée et la table des fragments est restaurée, ainsi votre programme aura encore accès à toute la mémoire disponible.

ExecDos comporte une autre caractéristique inhabituelle, son troisième paramètre est l'adresse d'une routine (écrite par vous) qui sauvegardera et restaurera l'écran et affichera des messages pour l'utilisateur. Cela vous donne toute la souplesse d'une routine écrite spécialement pour un programme donné, sans diminuer l'intérêt d'une routine appartenant à une bibliothèque traditionnelle.

Notez que la plupart des autres routines de la bibliothèque de *Turbo Professionnel* contiennent également des routines pour accéder au DOS et au BIOS. Si TPDOS ne peut pas répondre à vos besoins, vous avez de grandes chances de trouver satisfaction dans d'autres unités. TPCRT contient une grande variété de routines pour manipuler le curseur, écrire dans une mémoire vidéo, lire à partir d'une mémoire vidéo et lire le clavier. TPINT et TPTSR contiennent des routines pour manipuler des vecteurs d'interruption et écrire des programmes TSR. TPCHAINE contient des routines pour travailler avec des noms de fichiers DOS et ainsi de suite. Pour une liste complète des procédures et des fonctions de la bibliothèque *Turbo Professionnel,* voir l'index des procédures et fonctions à la fin de la documentation.

La nouvelle unité de *Turbo Professionnel* de 5.0, TPCLPLUS est regroupé avec l'unité TPDOS car sa principale fonction est l'interfaçage avec le BIOS. Cependant, contrairement à TPDOS, TPCLPLUS ne contient pas beaucoup de procédures et de fonctions. Elle met en oeuvre deux gestionnaires d'interruption pour trapper tous les appels système qui demandent des entrées clavier. Ils vous permettent d'écrire facilement des programmes qui supportent les claviers étendus (les nouveaux claviers avec 12 touches de fonction sur la partie supérieure) et permettent aux anciens claviers d'émuler des claviers étendus, lorsque cela est possible.

TPDOS : Appels au DOS ou au BIOS

Cette partie décrit tous les types, variables, constantes, procédures et fonctions qui se trouvent dans la partie interface de l'unité TPDOS.

Types

```
TypeCodeAction = (ExecSauveEcran, ExecMontreMem,
                  ExecPauseApresRun, ExecRestaureEcran);
```

Codes passés par *ExecDos* à une *RoutineUtilisateur*. La valeur *ExecSauveEcran* est passé en premier au moment de la sauvegarde de l'écran (en utilisant *SauveFen* par exemple). Remarquez que le buffer écran à utiliser doit avoir été alloué avant d'appeler *ExecDos*. La valeur *ExecMontreMem* est passé juste avant d'appeler DOS. En même temps, un deuxième paramètre (un mot) est passé, indiquant le total approximatif de mémoire disponible (en KiloOctets). La valeur *ExecPauseApresRun* est passé après le retour de l'appel à *Exec*. Vous pourriez utiliser cette facilité pour questionner l'utilisateur "appuyez sur une touche ..." et éviter d'effacer le contenu de l'écran avant que l'utilisateur ait pu le lire. La valeur *ExecRestaureEcran* est passé immédiatement après vous avoir donné la possibilité de restaurer l'écran.

Variables

```
PtrResultEs : Pointer;
```

Si ce pointeur vaut Nil (valeur par défaut), il doit pointer sur une routine telle que *ResultInt24* qui renvoie un code d'erreur d'entrée/sortie de type Word.

Constantes

```
MinEspacePourDos : Word = 20000;
```

Nombre minimum d'octets pour que le DOS puisse s'exécuter. Si *ExecDos* ne peut pas libérer suffisamment de place pour le DOS avant l'appel, elle se terminera anormalement.

```
MargeSecuritePile : Word = 1000;
```

Utilisée par *ExecDos* lors du déplacement de la liste d'espace libre vers le bas de la mémoire. Si *SizeOf(table des fragments) + MargeSecuritePile <= SP* (pointeur de pile actuel), la table des fragments sera enregistrée en bas de la pile, sinon sur le tas.

```
StdIn = 0; StdOut = 1;
StdErr = 2; StdPrn = 4;
```

Codes d'identification standard des fichiers DOS (entrée standard, sortie standard, sortie d'erreur standard et imprimante standard).

AnalyseChemin

Déclaration

```
function AnalyseChemin(var CheminEnt, CheminRech : string;
                       CheminEntete : string) : Boolean;
```

Objet

Elimine les blancs du nom de chemin entré, renvoie un nom de chemin global et un nom d'en-tête de chemin correct.

Commentaires

Cette routine complète les routines *FindFirst* et *FindNext* de l'unité standard DOS. *AnalyseChemin* analyse un masque de fichiers (*CheminEnt*) exactement comme le ferait une commande *Copy* du DOS, renvoie une chaîne passée à *FindFirst(CheminRech)* et une autre chaîne contenant simplement le chemin à rechercher. Les blancs de tête et de fin sont éliminés et tous les caractères sont convertis en majuscules. Par exemple, étant donné la chaîne ' \dos ', *AnalyseChemin* renverra '\DOS*.*' dans *CheminRech* et '\DOS\' dans *CheminEntete*. Elle renvoie "False" si *CheminEnt* ne peut pas être analysé.

AnnuleImpFich

Déclaration

```
procedure AnnuleImpFich(MasqueFich : string);
```

Objet

Supprime les fichiers qui correspondent au masque passé dans *MasqueFich.*

Commentaires

Cette routine vous permet de supprimer des fichiers de la file d'attente de PRINT.COM. *MasqueFich* est un masque de fichier DOS valide et peut inclure des jokers (? et *). Si le fichier en cours d'impression correspond à ce masque, DOS arrêtera immédiatement l'impression de ce fichier et passera au fichier suivant dans la file.

Exemple

```
AnnuleImpFich('*.DOC');
```

Annule l'impression de tous les fichiers avec l'extension 'DOC'

```
var
   FichEnCours : string[64];
begin
   Write('Fichier à imprimer : ');
   ReadLn(FichEnCours);
   if LanceImpFich(FichEnCours) <> 0 then {Erreur};
   {...Suite...}
   AnnuleImpFich(FichEnCours);
end.
```

Arrête l'impression en cours.

Voir aussi

AnnuleImpTousFichs (TPDOS)

AnnuleImpTousFichs

Déclaration

```
procedure AnnuleImpTousFichs;
```

Objet

Supprime tous les fichiers de la file d'attente de PRINT.COM.

Commentaires

Toutes les opérations d'impression sont immédiatement arrêtées. L'utilitaire PRINT ne terminera pas la sortie du fichier en cours.

Voir aussi

AnnuleImpFich (TPDOS)

CodesIdentFichs

Déclaration

```
function CodesIdentFichs : Byte;
```

Objet

Renvoie le nombre de codes d'identification de fichiers disponibles.

Commentaires

Cette routine essaie simplement d'ouvrir au plus 20 fichiers et renvoie le nombre de fichiers ouverts réellement. Tous les fichiers qui ont été ouverts seront fermés à la sortie. Cette routine est destinée principalement à des opérations de mise au point.

N'oubliez pas que cette fonction utilise à peu près 3K octets sur la pile.

Exemple

```
{$IFDEF Debug}
WriteLn(CodesIdentFichs,' identificateurs de fichiers sont
        disponibles');
{$ENDIF}
```

Utilisation typique de la fonction *CodesIdentFichs* dans une instruction de mise au point qui affiche le nombre de codes d'identification de fichiers disponibles.

CodesIdentFichsOuverts

Déclaration

```
function CodesIdentFichsOuverts(NombreUnites : Boolean);
```

Objet

Renvoie le nombre de fichiers ouverts par un programme.

Commentaires

Cette routine renvoie le nombre de fichiers que votre programme a ouvert. Le paramètre *NombreUnites* indique si les unités standards DOS (celles dont les codes d'identification sont dans l'intervalle *StdIn..StdPrn*) doivent être incluses dans le total. Le nombre renvoyé sera toujours dans l'intervalle 0..20 si *NombreUnites* vaut "True", ou 0..15 s'il vaut "False". N'oubliez pas qu'il n'est pas garanti que les 20 codes d'identification de fichiers soient disponibles dans votre programme sauf si "FILES=20" (ou un nombre supérieur à 20) apparaît dans le fichier CONFIG.SYS de l'utilisateur. La valeur par défaut de FILES est seulement de 8, malheureusement. Cette routine est destinée principalement pour la mise au point.

Exemple

```
{$IFDEF Debug}
WriteLn(CodesIdentFichsOuverts(True),' fichiers sont maintenant
        ouverts');
{$ENDIF}
```

Utilisation typique de *CodesIdentFichsOuverts,* dans une instruction de mise au point qui affiche le nombre de fichiers qui sont réellement ouverts.

EcritSecteursDisque

Déclaration

```
function EcritSecteursDisque(Lecteur, PremSect, NbSects : Word;
                             var Buf) : Boolean;
```

Objet

Ecrit des secteurs absolus sur le disque. *Utilisez cette routine avec beaucoup de précautions !*

Commentaires

Cette routine est la contrepartie de la routine *LitSecteursDisque* et doit toujours être utilisée en relation avec elle. Utilisé conjointement, elles vous permettent de lire dans un buffer des secteurs individuels, de modifier le contenu du buffer puis de réécrire les modifications sur disque.

Le numéro de lecteur est compris entre 0 (=A:) et 25 (=Z:). Pour convertir une lettre de lecteur en un numéro de lecteur, utilisez la formule suivante :

```
NumeroLecteur = Ord(UpCase(LettreLecteur)) -Ord ('A');
```

PremSect est le secteur à lire (le premier secteur du disque a pour numéro 0). *NbSects* est le nombre de secteurs à lire. *Buf* est le buffer dans lequel les données sont lues. Il doit avoir au moins une taille égale à *NbSects * OctetsParSecteur* (la routine *LitInfoDisque* peut être utilisée pour déterminer *OctetsParSecteur*).

Si la routine *EcritSecteursDisque* se termine bien, la valeur "True" est retournée. Sinon, elle renverra "False" et chargera *DosError* (une variable globale de l'unité DOS) avec un code d'erreur. Voir les explications de la routine *LitSecteursDisque* pour la liste de tous les codes des erreurs possibles.

Remarque : *EcritSecteursDisque* supporte seulement les versions 3.30 du DOS et antérieures. Toute version supérieure à 3.30 renverra "False" et *DosError* sera positionnée à $FFFF. En cas de modification, le fichier ALIRE de la disquette DEMO contiendra les dernières informations.

Exemple

```
var
   Lecteur : Byte;
   Buffer : array[1..2048] of Byte;
begin
   Lecteur := Ord(LecteurParDefaut) - Ord('A');
   if not LitSecteursDisque(Lecteur, 0, 1, Buffer) then begin
      WriteLn('Erreur ', DosError, ' pendant la lecture sur ',
               LecteurParDefaut);
      DosError := 0;
      Exit;
   end;
   ...
   if not EcritSecteursDisque(Lecteur, 0, 1, Buffer) then begin
      WriteLn('Erreur ', DosError, ' pendant l''écriture sur ',
                  LecteurParDefaut);
      DosError := 0;
      Exit;
   end;
end.
```

Voir aussi

LitInfoDisque (TPDOS) LitSecteursDisque (TPDOS)

ExecDos

Déclaration

```
function ExecDos(Commande : string; Deuxieme : Boolean;
                 RoutineUtl : Pointer) : Integer;
```

Objet

Exécute toute commande du DOS.

Commentaires

ExecDos compresse le tas temporairement afin de libérer assez de place pour le programme en cours d'exécution. Le tas est restauré à sa valeur d'origine après l'appel de *Exec*.

ExecDos vous évite de restaurer les vecteurs d'interruption avant d'exécuter un autre programme. Lorsqu'elle est compilée sous *Turbo Pascal* 4.0, elle restaure automatiquement les interruptions $23(Control Break du DOS) et $24(Erreur critique du DOS) avant d'appeler Exec, et les restaure ensuite à leur valeur précédente. Lorsqu'elle est compilée sous *Turbo Pascal* 5.0, elle utilise la routine *SwapVectors* de l'unité DOS pour restaurer temporairement tous les vecteurs utilisés par l'unité SYSTEM.

Commande est le nom d'un programme à exécuter ou une chaîne vide (qui signifie "appel d'une commande du DOS"). *Deuxieme* indique si une seconde copie de COMMAND.COM est chargée pour traiter l'appel de *Exec*. Si le paramètre est "False", la commande doit être un chemin complet de programme à exécuter. (Voir l'entrée de la fonction *ExisteSurChemin*). *RoutineUtl* pointe sur une routine utilisateur qui sera appelée pour afficher des informations, etc. Elle doit être de la forme

```
procedure Routine(CodeAction : TypeCodeAction; Param : Word);
```

et elle devrait être compilée avec l'option FAR (en utilisant la directive de compilation {$F+}). Pour des explications sur les codes actions, voir *TypeCodeAction*. Si vous n'avez pas besoin de cette possibilité, passez un pointeur avec la valeur Nil.

Les codes renvoyés par *ExecDos* sont :

- 0 Bien terminé
- -1 Mémoire insuffisante pour enregistrer la table des fragments de Turbo
- -2 Erreur de bloc avant l'appel à Exec
- -3 Erreur de bloc après l'appel à Exec (erreur critique)
- -4 Mémoire insuffisante pour exécuter la commande DOS

sinon code erreur du DOS

Note : Vous ne devez pas utiliser *ExecDos* pour exécuter un programme TSR. *ExecDos* utilise une partie de la mémoire allouée pour vos programmes en espérant pouvoir la récupérer plus tard. Si cela n'est pas possible, par exemple si vous essayez d'exécuter un programme TSR, elle retournera un code d'erreur (probablement -3).

Exemples

```
Code := ExecDos('C:\TURBO\TPC.EXE '+NomProg, False, Nil);
```

Appelle Turbo pour compiler *NomProg* ; puisque le paramètre *Deuxieme* vaut "False", le nom complet de chemin pour TPC.EXE doit être spécifié.

```
Code := ExecDos('CREEMENU', True, Nil);
```

Exécute MENU. Une deuxième copie de COMMAND.COM sera chargée pour trouver et exécuter le programme.

```
var
   P         : Pointer;
   TailleBuf : Word;
   Code      : integer;

{$F+}
procedure RoutineUtl(CodeAction : TypeCodeAction; Param : Word);
var
   Touche : Word;
begin
   case CodeAction of
      ExecSauveEcran  :              {n'alloue pas de Buffer écran,
                                      sauvegarde l'écran}
      if not SauveFen(1,1,LargeurEcran,Succ(HauteurEcran), False,P)
         then                        {ne peut pas échouer --
                                      le Buffer est déjà alloué};
      ExecMontreMem  :               {montre la mémoire disponible
                                      pour le shell}
         writeln('Approximativement ',Param,' K octets disponibles');
      ExecPauseApresRun :            {soyez sûr que l'écran ne soit pas
                                      effacé trop vite}
         begin
            write('Tapez une touche pour continuer...');
            Touche :=LitMotTouche;
         end;
      ExecRestaureEcran:             {rétablit l'écran, le buffer écran
                                      n'est pas libéré}
        RestaureFen(1,1,LargeurEcran,Succ(HauteurEcran),False,P);
   end;
end;
{$F-}
```

```
begin
   ...
   {alloue un Buffer écran}
   TailleBuf := 2*LargeurEcran*Succ(HauteurEcran);
   GetMem(P,TailleBuf);
   Code := ExecDos('',True,@RoutineUtl);
   FreeMem(P,TailleBuf);
end.
```

Voir aussi

Déclarations (TPDOS)
ExisteSurChemin (TPOS)
MinEspacePourDos (TPDOS)
MargeSecuritePile (TPDOS)

ExisteSurChemin

Déclaration

```
function ExisteSurChemin(NomF : String;
                         var NomComplet : String) : Boolean;
```

Objet

Renvoie "True" si *NomF* est trouvé et renvoie le nom complet du chemin de ce fichier.

Commentaires

Contrairement à *FichExiste*, cette routine peut retrouver des fichiers qui ne sont pas dans le répertoire courant. Si *NomF* contient un identificateur de lecteur et/ou de chemin, seuls le lecteur et le répertoire spécifiés sont analysés. Mais si un nom de fichier seulement est spécifié, *ExisteSurChemin* recherchera dans cet ordre : répertoire courant, répertoire du programme exécutable (sous DOS 3.x seulement), tous les répertoires spécifiés par la commande PATH du DOS. Cette routine est particulièrement utile pour localiser tout fichier de configuration, de données, ou tout autre fichier dont votre programme peut avoir besoin. Comme *FichExiste*, *ExisteSurChemin* n'accepte pas les jokers et ne peut pas localiser des fichiers accessibles en lecture seulement. La solution à ce problème est la même que pour *FichExiste*, utilisez une routine telle que :

```
function CeFichierExisteQuelquePart(NomF : String;
                                    var NomComplet : string) : boolean;
var SauveMode : Byte;
begin
   SauveMode := FileMode;
   FileMode := 0;
   CeFichierExisteQuelquePart := ExisteSurChemin(NomF, NomComplet);
   FileMode := SauveMode;
end;
```

Exemples

```
Trouve := ExisteSurChemin('*.DOC', NomComplet);
```

Incorrect! ExisteSurChemin n'accepte pas les caractères génériques.

```
Trouve := ExisteSurChemin('C:\AUTOEXEC.BAT', NomComplet);
```

ExisteSurChemin cherchera 'AUTOEXEC.BAT' seulement sur 'C:\'.

```
Trouve := ExisteSurChemin('AUTOEXEC.BAT', NomComplet);
```

ExisteSurChemin fera une recherche complète pour 'AUTOEXEC.BAT'.

Voir aussi

FichExiste (TPDOS)

FichExiste

Déclaration

```
function FichExiste(NomF : string) : Boolean;
```

Objet

Renvoie "True" si NomF est trouvé.

Commentaires

Puisque *FichExiste* utilise la routine standard *Reset* de Turbo, elle ne pourra généralement pas trouver les fichiers qui sont accessibles en lecture seulement. En cas de problèmes, vous pouvez utiliser une routine d'une couche supérieure qui réinitialisera la variable *FileMode* de Turbo (valeur par défaut = 2) avant d'appeler *FichExiste.*

```
function CeFichierExiste(NomFich : String) : Boolean;
var
   SauveMode : Byte;
begin
   SauveMode := FileMode;
   FileMode := 0;
   CeFichierExiste := FichExiste(NomFich);
   FileMode := SauveMode;
end;
```

Si vous travaillez sur un réseau, vous pouvez utiliser des valeurs différentes de 2 ou 0.

FichExiste n'accepte pas les jokers et recherche les fichiers seulement dans le répertoire courant ou dans le répertoire spécifié par *NomF*.

Exemples

```
Trouve :=  FichExiste('*.DOC');
```

Incorrect ! FichExiste n'accepte pas les caractères génériques.

```
Trouve :=  FichExiste('C:\AUTOEXEC.BAT');
```

FichExiste cherchera 'AUTOEXEC.BAT' sur 'C:\'.

```
Trouve :=  FichExiste('AUTOEXEC.BAT');
```

FichExiste cherchera 'AUTOEXEC.BAT' sur le répertoire actuel.

Voir aussi

ExisteSurChemin (TPDOS)

FinEtatImp

Déclaration

```
procedure FinEtatImp;
```

Objet

Libère le spooler de la procédure *LitEtatImp*.

Commentaires

Réactive le spooler d'impression après un appel à *LitEtatImp* pour l'impression.

Voir aussi

LitEtatImp (TPDOS)

HorlogeMs

Déclaration

```
function HorlogeMs : LongInt;
```

Objet

Renvoie l'heure du jour en millisecondes écoulées depuis minuit.

Commentaires

Cette routine offre un moyen pratique de calculer le temps d'exécution d'un programme (en millièmes de secondes).

Veuillez noter que la précision du résultat est limitée par l'horloge système à environ 55 millisecondes (1/18ème de secondes). Si vous avez besoin d'un chronométrage plus précis, utilisez l'unité TPTIMER qui se trouve dans le sous-répertoire BONUS de la disquette SRC3.

Exemple

```
var
   Debut, Fin : LongInt;
begin
   Debut := HorlogeMS;
   {...Corps du programme principal...}
   Fin := HorlogeMS;
   WriteLn('Temps écoulé : ', (Fin-Debut) / 1000.0);
end.
```

IdentEstConsole

Déclaration

```
function IdentEstConsole(Identificateur : Word) : Boolean;
```

Objet

Renvoie "True" si l'identificateur de périphérique est la console.

Commentaires

IdentEstConsole permet, entre autres choses, de déterminer si les entrées/sorties d'un programme doivent être redirigées ou non. Cette information peut servir pour écrire un *filtre* tel que BTRI.PAS qui devra savoir si l'utilisateur a redirigé les entrées/sorties comme demandé.

Exemple

```
if IdentEstConsole(StdIn) then
   WriteLn('L''entrée doit être redirigée depuis un fichier');
if IdentEstConsole(StdOut) then
   WriteLn('La sortie doit être redirigée vers un fichier');
```

Pour un autre exemple, voir la routine *SortStr* dans BTRI.PAS.

InitDta

Déclaration

```
procedure InitDta(PtrDTA : Pointer);
```

Objet

Fait pointer la zone de transfert disque sur *PtrDTA*.

Commentaires

Le DOS utilise la zone de transfert disque pour un certain nombre d'opérations sur disque et, notamment, pour les recherches dans les répertoires. Une technique courante est de faire pointer la zone de transfert disque du DOS sur une région recouverte par un enregistrement Pascal. Une fois que le DOS a écrit les informations du répertoire dans cette zone, les rubriques de l'enregistrement Pascal fournissent un moyen pratique pour analyser les données.

En fait, c'est la technique utilisée par la routine *FindFirst* de l'unité DOS. *FindFirst* et *FindNext* offrent des moyens pratiques de lire le répertoire du DOS mais elles ne sont pas aussi efficaces que le programme CHERCHE qui recherche récursivement dans les répertoires.

InitDta appelle la fonction DOS $1A. Habituellement, le paramètre doit être un pointeur sur *SearchRec* (défini dans l'unité DOS).

Exemple

```
var
   SR : SearchRec;
begin
   InitDta(@SR); {Fait pointer le DTA sur SR}
end.
```

Voir aussi

LitDta (TPDOS)

InitVerif

Déclaration

```
procedure InitVerif(On : Boolean);
```

Objet

Active et désactive la vérification des opérations d'écriture sur disque.

Commentaires

InitVerif sert le plus souvent dans des programmes qui manipulent des données vitales. Lorsque VERIFY est ON, le DOS contrôle automatiquement l'intégrité de toutes les données écrites sur le disque.

Exemple

```
if not VerifOn then InitVerif(True);
```

Active la vérification.

```
if VerifDoitEtreOff then InitVerif(False);
```

Désactive la vérification pour écrire plus vite sur disque.

Voir aussi

VerifOn (TPDOS)

LanceImpFich

Déclaration

```
function LanceImpFich(NomFich : string) : Byte;
```

Objet

Cette procédure soumet un fichier à PRINT.COM.

Commentaires

Cette routine permet de soumettre des fichiers pour les impressions simultanées sous DOS 3.x à condition que PRINT.COM soit installé. Avant de l'appeler, vous devrez vérifier que ces conditions sont remplies en appelant *PrintPresent*.

Si les conditions sont remplies, *LanceImpFich* renvoie 0, sinon l'un des codes d'erreur suivants est retourné :

1	Fonction non valide
2	Fichier non trouvé
3	Chemin non trouvé
4	Trop de fichiers ouverts
5	Accès refusé
8	File d'impression pleine
9	Occupé
12	Nom trop long
15	Lecteur non valide

Exemple

```
if not PrintPresent then
   WriteLn('PRINT.COM n''est pas en mémoire')
else
   if LanceImpFich('\AUTOEXEC.BAT') <> 0 then
      WriteLn('Impossible de lancer l''impression');
```

Voir aussi

PrintPresent (TPDOS)

LecteurParDefaut

Déclaration

```
function LecteurParDefaut : Char;
```

Objet

Renvoie le lecteur par défaut en lettres majuscules.

Commentaires

Renvoie un caractère dans l'intervalle 'A'..'Z'. Pour convertir ce résultat en un numéro de lecteur, vous pourriez écrire

```
NumLecteur := Ord(LecteurParDefaut) - 64;
```

Exemple

```
writeln('Lecteur actuel : ', LecteurParDefaut);
```

Afficherait 'Lecteur actuel : C' si C: est le lecteur actuel.

Voir aussi

NbLecteurs (TPDOS)

SelectLecteur (TPDOS)

LitDta

Déclaration

```
procedure LitDta(var PtrDTA : pointer);
```

Objet

Renvoie le pointeur sur l'adresse de transfert disque du DOS.

Commentaires

LitDta renvoie un pointeur sur l'adresse de transfert du disque actif. Cette routine est importante lorsque l'adresse de transfert du disque doit être temporairement redirigée avec *InitDta*, puis restaurée. Ceci est très important pour les programmes résidents en mémoire qui doivent accéder au disque via le DOS. Cependant, vous n'aurez pas à appeler *LitDta/InitDta* dans ce cas, puisque les routines "popup" de *Turbo Professionnel* le font automatiquement.

Exemple

```
var
   AncDta : Pointer;
   SR     : SearchRec;
begin
   LitDta(AncDta);  {Sauve le DTA actuel}
   InitDta(@SR);    {Fait pointer le DTA sur un masque de recherche}
   {-Traitement sur des fichiers ou des répertoires}
   InitDta(AncDta); {Restaure le DTA d'origine}
end.
```

Voir aussi

InitDta (TPDOS)

LitEnvironnement

Déclaration

```
function LitEnvironnement(ChaineRech : string) : string;
```

Objet

Renvoie une chaîne d'environnement.

Commentaires

Cette routine permet de localiser des chaînes relatives aux variables d'environnement tel que 'COMSPEC', 'PATH' et 'PROMPT'. La recherche se fait sans tenir compte des majuscules et minuscules et vous n'avez pas à spécifier '=' à la fin de la chaîne de recherche.

Exemples

```
writeln(LitEnvironnement('COMS'));
```

Correct, seulement si le nom complet de la variable est 'COMS'. Ceci ne localisera pas pas 'COMSPEC'.

```
writeln(LitEnvironnement(' COMSPEC  '));
```

Incorrect ! Les espaces aux extrémités doivent être supprimés.

```
WriteLn(LitEnvironnement('comspec='));
WriteLn(LitEnvironnement('comspec'));
WriteLn(LitEnvironnement('COMSPEC='));
WriteLn(LitEnvironnement('COMSPEC'));
```

Toutes ces instructions sont correctes et renvoient 'C:\COMMAND.COM'.

LitEtatImp

Déclaration

```
function LitEtatImp(var PtrQ : Pointer) : Byte;
```

Objet

Arrête l'impression, renvoie le code d'erreur en cours, place le pointeur sur la file des noms de fichiers dans la variable *PtrQ*.

Commentaires

Cette routine renvoie un pointeur sur une suite de chaînes ASCIIZ qui représentent les noms des fichiers dans la file d'impression. Chaque entrée a une taille de 64 octets et la fin de la file est marquée par le nom commençant par un caractère nul. L'impression est suspendue dès l'appel à *LitEtatImp* jusqu'à ce que vous appeliez *FinEtatImp*. Remarquez que vous pouvez facilement convertir les chaînes ASCIIZ en chaînes normales en utilisant la routine *AscEnChn* de TPASCIIZ. *LitEtatImp* renvoie la valeur 0 si l'opération est bien terminée.

Exemple

```
var
   P : Pointer;
   AP : AsciizPtr absolute P;
   POffset : Word absolute P;
begin
   if LitEtatImp(P) = 0 then
      while Char(P^) <> #0 do begin        {vérification pour zéro}
         writeln(AscEnChn(AP^));           {affiche l'entrée en cours}
         Inc(POffset,64);                  {pointe l'entrée suivante}
      end;
   FinEtatImp;      {libère le spooler d'imprimante}
end;
```

Voir aussi

AscEnChn (TPASCIIZ)

FinEtatImp (TPDOS)

LitInfoDisque

Déclaration

```
function LitInfoDisque(Lecteur : Byte; var ClustersDisponibles,
                       TotalClusters, OctetsParSecteur,
                       SecteursParCluster : Word) : Boolean;
```

Objet

Renvoie des informations techniques sur le lecteur spécifié.

Commentaires

Cette routine renvoie des informations sur l'organisation d'un lecteur donné. Vous pouvez l'utiliser pour calculer la capacité totale d'un disque, le nombre d'octets disponibles, ou pour obtenir les données à utiliser en relation avec *LitSecteursDisque / EcritSecteursDisque.*

Lecteur est le numéro du lecteur concerné : 0 = lecteur par défaut, 1 = A, 2 = B, etc... Pour convertir une lettre de lecteur en un numéro de lecteur, utilisez la formule suivante :

```
NumeroLecteur : = Ord(UpCase(LettreLecteur)) -64;
```

La fonction renvoie "False" si le numéro de lecteur est incorrect, sinon, elle renvoie les valeurs suivantes : *ClustersDisponibles* (nombre de clusters inutilisés sur le disque), *TotalClusters* (nombre total de clusters sur le disque), *OctetsParSecteur* (nombre d'octets dans un secteur de disque donné, la plupart du temps égal à 512), et *SecteursParCluster* (nombre de secteurs dans un cluster, habituellement 1, 2, 4).

Exemple

```
var
   OctetsParCluster, CapaciteDisque, EspaceLibre : LongInt;
   ClustersDisponibles, TotalClusters,
   OctetsParSecteur,SecteursParCluster : Word;
begin
   if LitInfoDisque(0, ClustersDisponibles, TotalClusters,
         OctetsParSecteur, SecteursParCluster) then begin
      OctetsParCluster := LongInt(SecteursParCluster) * LongInt(OctetsParSecteur);
      CapaciteDisque:= LongInt(TotalClusters)*OctetsParCluster;
      EspaceLibre := LongInt(ClustersDisponibles)*OctetsParCluster;
      writeln('Informations sur le lecteur ',LecteurParDefaut,':');
      writeln('Octets par secteur: ',OctetsParSecteur);
      writeln('Octets par cluster: ',OctetsParCluster);
      writeln('Capacité du disque: ',CapaciteDisque);
      writeln('Octets disponibles: ',EspaceLibre);
   end;
end.
```

Voir aussi

LitSecteursDisque (TPDOS) EcritSecteursDisque (TPDOS)

LitModeFich

Déclaration

```
function LitModeFich(NomF : string, var Attr : Word) : Byte;
```

Objet

Lit les attributs d'un fichier.

Commentaires

Cette routine permet d'utiliser la routine de niveau supérieur *GetFAttr* de Turbo plus facilement. Elle renvoie l'attribut de *NomF* dans *Attr* et le code erreur du DOS (DosError) en résultat de la fonction. L'attribut est codé comme suit :

Bit	Valeur	Signification
0	$01	Accès en lecture seulement
1	$02	Caché
2	$04	Système
3	$08	Label de volume
4	$10	Sous-répertoire
5	$20	Archive

Pour voir si un fichier est caché, vous pourriez par exemple utiliser le test suivant :

```
Cache := (Attr and $02) <> 0;
```

Exemple

```
var
   NomF : String;
   Attr : Word;
begin
   Write('Entrez le nom du fichier : ');
   Readln(NomF);
   if LitModeFich(NomF, Attr) <> 0 then
      WriteLn('Fichier introuvable')
   else begin
      Write(NomF, ' est un ');
      if (Attr and $10) <> 0 then
         WriteLn('répertoire')
      else
      WriteLn('fichier');
   end;
end.
```

LitSecteursDisque

Déclaration

```
function LitSecteursDisque(Lecteur, PremSect, NbSects:Word;
                           var Buf) : Boolean;
```

Objet

Lit des secteurs absolus sur disque.

Commentaires

Cette routine permet d'accéder à des secteurs individuels sur disque et d'analyser des parties inutilisées du disque. Le numéro de lecteur est compris entre 0 (= A:) et 25 (= Z:). Pour convertir une lettre de lecteur en un numéro de lecteur, utilisez la formule suivante :

```
NumeroLecteur := Ord(UpCase(LettreLecteur)) - Ord('A');
```

PremSect est le secteur à lire (premier secteur du disque = 0). *NbSects* est le nombre de secteurs à lire. *Buf* est le buffer dans lequel les données seront lues. Il doit avoir une taille au moins égale à *NbSects* * *OctetsParSecteur* (voir *LitInfoDisque*). Si la routine *LitSecteursDisque* est interrompue, elle renverra "False" et placera un code d'erreur dans *DosError*, interprété comme suit :

Octet de poids fort (code erreur généré par le BIOS) :

128	Temps dépassé: le lecteur n'a pas répondu
64	Opération de recherche interrompue
32	Contrôleur en panne
16	CRC incorrect (contrôle de redondance cyclique) erreur de parité
8	Erreur d'accès direct à la mémoire (DMA)
4	Secteur non trouvé
3	Disquette protégée en écriture
2	Marque d'adresse incorrecte, l'identificateur de secteur est incorrect ou non trouvé.

Octet de poids faible (code erreur généré par DOS) :

12	Panne générale
11	Erreur en lecture
10	Erreur en écriture
8	Secteur non trouvé
7	Périphérique inconnu, format disque non reconnu
6	Erreur en recherche (seek)
4	CRC incorrect
2	Lecteur non prêt
1	Unité inconnue : numéro de lecteur incorrect
0	Tentative d'écriture sur une disquette protégée.

Remarque : *LitSecteursDisque* supporte actuellement seulement les versions DOS 3.30 et antérieures. Si vous possédez une version du DOS supérieure à 3.30 elle renverra "False" et *DosError* aura la valeur $FFFF. Si cette limite est augmentée, le fichier ALire de la disquette DEMO contiendra des informations à ce sujet.

Exemple

```
   var
      Lecteur : Byte;
      Buffer  : array[1..2048] of byte;
   begin
      Lecteur := Ord(LecteurParDefaut) - Ord('A');
      if not LitSecteursDisque(Lecteur, 0, 1, Buffer) then begin
         Writeln('Erreur ',DosError,' survenue pendant une lecture sur ',
                  LecteurParDefaut);
         DosError := 0;
         Exit;
      end;
end.
```

Voir aussi

LitInfoDisque (TPDOS)

EcritSecteursDisque (TPDOS)

ModifBloc

Déclaration

```
function ModifBloc(var Paragraphes : Word ) : Boolean;
```

Objet

Modifie la taille du bloc alloué en mémoire pour ce programme.

Commentaires

Cette routine destinée à l'origine pour les utilisations internes avec *ExecDos* vous permet d'augmenter ou de diminuer la taille de la mémoire allouée à votre programme. *Paragraphes* est la nouvelle taille du bloc de mémoire demandée. Si la demande ne peut être satisfaite, *ModifBloc* renverra "False" et *Paragraphes* contiendra le nombre de paragraphes disponibles.

Très important : si vous modifiez le bloc mémoire d'un programme, vous devez être certain que la table des fragments de Turbo est correcte. La routine *ExecDos* dans TPDOS.PAS vous montrera comment modifier en toute sécurité le bloc mémoire.

Exemples

Voir la routine *ExecDos* dans TPDOS pour des exemples.

Voir aussi

ExecDos (TPDOS)

NbLecteurs

Déclaration

```
function NbLecteurs : Byte;
```

Objet

Renvoie le nombre de lecteurs logiques.

Commentaires

Sous DOS 3.0 ou les versions supérieures, la valeur retournée sera toujours au moins 5 puisque DOS 3.x supporte les lecteurs SUBST. Sous DOS 2.x, le nombre réel de lecteurs sera retourné.

Exemple

```
WriteLn('Nombre de lecteurs, selon la Version DOS : ', NbLecteurs);
```

OuvreUniteStd

Déclaration

```
function OuvreUniteStd(var F : Text; IdentStd : Word) : Boolean;
```

Objet

Affecte un fichier texte à une unité standard du DOS : 0, 1, 2 ou 4.

Commentaires

Cette routine permet d'accéder facilement aux unités standard d'entrée, de sortie, de sortie d'erreur et d'impression. *F* est un fichier texte non ouvert et *IdentStd* est égal à 0 (StdIn), 1 (StdOut), 2 (StdErr), ou 4 (StdPrn). De plus, *OuvreUniteStd* permet d'écrire sur une unité standard de sortie dans des programmes qui utilisent CRT ou TPCRT. Elle renvoie "False" seulement si *IdenStd* est un numéro incorrect.

Exemple

```
var
   SortieStd : Text;
begin
   if not OuvreUniteStd(SortieStd, StdOut) then
      {Erreur};
end.
```

PosTexte

Déclaration

```
function PosTexte(var F : Text) : LongInt;
```

Objet

Renvoie la position actuelle du pointeur logique de fichier.

Commentaires

PosTexte récupère la position actuelle du pointeur physique de fichier puis adapte cette valeur pour tenir compte du contenu actuel du buffer du fichier texte. Le résultat est la position actuelle du pointeur logique de fichier, la position que le pointeur physique de fichier aurait s'il n'y avait pas de buffer. *PosTexte* renvoie -1 en cas d'erreur d'entrée/sortie.

Voir l'entrée de *RechTexte* en exemple.

Voir aussi

RechTexte (TPDOS)

PrintPresent

Déclaration

```
function PrintPresent : Boolean;
```

Objet

Renvoie "True" si PRINT.COM est installé.

Commentaires

Toutes les routines d'accès au spooler d'impression du DOS (PRINT.COM) fonctionnent avec la version DOS 3.0 ou supérieure. Si le programme fonctionne sous une version antérieure, *PrintPresent* renverra toujours "False". Sinon, elle renverra "False" seulement si le spooler d'impression n'est pas résident.

Exemple

```
if not PrintPresent then
   WriteLn('PRINT.COM n''est pas en mémoire');
```

Voir aussi

AnnuleImpTousFichs (TPDOS)
AnnuleImpFich (TPDOS)
FinEtatImp (TPDOS)
LitEtatImp (TPDOS)
LanceImpFich (TPDOS)

RechTexte

Déclaration

```
function RechTexte(var F : Text; Cible : LongInt) : Boolean;
```

Objet

Recherche dans un fichier texte ouvert en lecture.

Commentaires

Cette routine permet d'exécuter des accès directs sur des fichiers textes ouverts en lecture. *RechTexte* renvoie "False" si le fichier n'est pas ouvert en lecture ou si une erreur d'entrée/sortie se produit.

Remarquez que *RechTexte* vide quelquefois le buffer du fichier texte, aussi pour des raisons de performances, il est très prudent de l'utiliser avec des fichiers comportant de grands buffers créés avec *SetTextBuf*. *RechTexte* évitera cela toutes les fois que c'est possible : si la donnée à rechercher est déjà dans le buffer, elle repositionnera simplement le pointeur du buffer du fichier plutôt que de rechercher dans le fichier physique.

Exemple

```
var
   DestAcces : LongInt;
   F : Text;
begin
   {...Ouverture de F...}
   Write('Entrez la position d''accès : ');
   ReadLn(DestAcces);
   if not RechTexte(F, DestAcces) then
      WriteLn('Erreur')
   else
      WriteLn('DestAcces : ', DestAcces, ' PosTexte : ', PosTexte(F));
end.
```

Voir aussi

PosTexte (TPDOS)

SelectLecteur

Déclaration

```
procedure SelectLecteur(Lecteur : Char);
```

Objet

Sélectionne le lecteur spécifié comme lecteur par défaut si cela est possible.

Commentaires

Le lecteur doit être une lettre comprise dans l'intervalle 'a'..'z' ou 'A'..'Z'. Si le lecteur est incorrect, *SelectLecteur* s'arrête.

Exemples

```
SelectLecteur('A');
SelectLecteur('b');
SelectLecteur('C');
```

Voir aussi

LecteurParDefaut (TPDOS) NbLecteurs (TPDOS)

TailleFichTexte

Déclaration

```
function TailleFichTexte(var F : Text) : LongInt;
```

Objet

Renvoie la taille du fichier texte *F*.

Commentaires

Cette routine permet de déterminer la taille actuelle de n'importe quel fichier texte ouvert en lecture ou en écriture. Le contenu du buffer du texte en cours est comptabilisé dans le résultat. *TailleFichTexte* renvoie -1 en cas d'erreur d'entrée/sortie.

Exemple

```
var
   TailleDeMonFichier : LongInt;
   F : Text;
begin
   {...Ouverture de F ...}
   TailleDeMonFichier := TailleFichTexte(F);
   if TailleDeMonFichier = -1 then
      WriteLn('Erreur')
   else
   WriteLn('Taille = ', TailleDeMonFichier);
end.
```

VerifOn

Déclaration

```
function VerifOn : Boolean;
```

Objet

Renvoie "True" si la vérification en écriture sur le disque est à 'ON'.

Commentaires

Permet de savoir si le DOS est en train de vérifier l'intégrité des données écrites sur le disque.

Exemple

```
Write('VERIFY est positionnée à ');
if VerifOn then
   WriteLn('On')
else
   WriteLn('Off');
```

Voir aussi

InitVerif (TPDOS)

VersionDos

Déclaration

```
function VersionDos : Word;
```

Objet

Renvoie le numéro de version du DOS.

Commentaires

Le résultat est en format BCD (Binary Coded Decimal), avec le numéro de la dernière version dans l'octet de poids fort et le numéro de la version de base dans l'octet de poids faible. Par exemple, sous DOS 3.1 le résultat serait $030A ($0A = 10), sous DOS 2.11 $020B ($0B = 11), sous DOS 3.3, $031E ($1E = 30) et ainsi de suite.

Exemples

```
if VersionDos < $0300 then
   Writeln('Ce programme nécessite un DOS 3.xx');
```

Test de DOS 3.x ou plus

```
if VersionDos > $0400 then
   Writeln('Ce programme n''a pas été testé avec un DOS 4.xx');
```

Test de DOS 4.00 ou plus

VideBufTexte

Déclaration

```
function VideBufTexte(var F : Text) : Boolean;
```

Objet

Vide le(s) buffer(s) d'un fichier texte.

Commentaires

Cette routine vide le buffer d'un fichier texte. Si le fichier a été ouvert en écriture, le contenu actuel du buffer est écrit sur disque et les buffers DOS sont vidés en utilisant les fonctions $45 (Code d'indentification de fichiers en double) et $3E (Ferme fichier) du DOS. S'il a été ouvert en lecture, le pointeur de fichier DOS est déplacé sur le prochain caractère à lire (par exemple, l'adresse qu'il aurait s'il n'y avait pas de buffer). *VideBufTexte* renvoie "False" en cas d'erreur d'entrée/sortie.

Exemple

```
var
   F : Text;
begin
   {...Ouverture de F...}
   if not VideBufTexte(F) then
      WriteLn('Erreur en vidant le Buffer du fichier')
   else
      WriteLn('Buffer du Fichier vidé');
end.
```

TPCLPLUS : Interface de clavier étendu

Bien que les claviers étendus (ceux qui possèdent 12 touches de fonction au lieu de 10) soient des équipements standard sur les nouvelles machines, il existe encore relativement peu de programmes qui les exploitent complètement. Il y a trois bonnes raisons pour cela. Premièrement, pour pouvoir lire toutes les touches d'un clavier étendu, vous devez utiliser des fonctions spéciales du BIOS. Deuxièmement, avant de pouvoir utiliser ces fonctions, vous devez être sûr qu'un clavier étendu est installé ; il n'existe pas de moyen standard de le savoir. Troisièmement, il est difficile d'écrire des programmes qui fonctionnent aussi bien avec des anciens claviers et des claviers étendus, puisque de nombreuses combinaisons de touches spéciales reconnues par un clavier étendu telles que (ALT)-(ESC) et (CTRL)-(TAB), ne sont pas reconnues par les anciens claviers.

TPCLPLUS résout ces problèmes en détectant et en activant un clavier étendu sur les machines qui en possèdent un, et en le simulant sur les machines qui n'en possèdent pas. Lorsque TPCLPLUS est installé, des codes de balayage étendus seront retournés par toutes les procédures de lecture du clavier : *ReadKey* (CRT/TPCRT), *LitMotTouche* (TPCRT), et toutes les routines qui utilisent le clavier standard du DOS et du BIOS. Vous n'aurez pas à appeler de routines puisque cette unité s'installe et se désinstalle automatiquement. Cette opération est complètement transparente aux programmes qui l'utilisent.

Mode de fonctionnement de TPCLPLUS

TPCLPLUS utilise deux gestionnaires d'interruption. Le premier gère l'interruption $16, interruption clavier du BIOS, et intercepte tous les appels destinés au BIOS nécessitant une entrée clavier. Si un clavier étendu est installé, le gestionnaire d'interruptions déroute tous les appels aux fonctions clavier standard ($0, $1, et $2) vers les fonctions du clavier étendu correspondantes ($10, $11, et $12).

Le deuxième gestionnaire d'interruption gère l'interruption $9, l'interruption matérielle générée chaque fois qu'une touche est pressée ou relâchée. Il contrôle certaines combinaisons de touches qui ne sont pas normalement retournées par le BIOS. Si l'une de ces combinaisons apparaît, il contourne le gestionnaire clavier du BIOS et insère le code de la touche appropriée dans le buffer du clavier (voir la table ci-dessous).

Codes des touches

La table qui suit résume les codes touches renvoyés lorsqu'une touche donnée ou une combinaison de touches est pressée alors que TPCLPLUS est activée. Tous les nombres sont en hexadécimal et indiquent la valeur qui sera renvoyée par un appel à *LitMotTouche* de TPCRT. L'octet de poids fort du mot est le code de balayage et l'octet de poids faible est le caractère Ascii. Si vous utilisez la fonction *ReadKey* (de CRT ou TPCRT) pour lire le clavier, et que le caractère retourné est #0 (représentant l'octet de poids faible du code de la touche), un deuxième appel à *ReadKey* renverra le code de balayage (octet de poids fort). Toutes les touches qui n'apparaissent pas dans la liste sont retournées normalement (voir le tableau des codes de balayage en appendice E du manuel *Turbo Pascal* 4.0 / Appendice C du guide de référence *Turbo Pascal* 5.0). Voir les symboles spéciaux qui apparaissent après certains codes de touches (paragraphe *Notes*).

	Normal	Shift	Ctrl	Alt
(F11)	8500%	8700%	8900%	8B00%
(F12)	8600%	8800%	8A00%	8C00%
(ESC)	011B	011B	011B	0100*
`	2960	297E	2960	2900*
(BACKSPACE)	0E08	0E08	0E7F	0E00*
(TAB)	0F09	0F00	9400*	A500*
{	1A5B	1A7B	1A1B	1A00*
}	1B5D	1B7D	1B1D	1B00*
\	2B5C	2B7C	2B1C	2B00*
;	273B	273A	----	2700*
'	2827	2822	----	2800*
(ENTER)	1C0D	1C0D	1C0A	1C00*
,	332C	333C	----	3300*
.	342E	343E	----	3400*
/	352F	353F	----	3500*
(INS)	5200	5230	9200*	A200#
(DEL)	5300	532E	9300*	A300#
(HOME)	4700	4737	7700	9700#
(END)	4F00	4F31	7500	9F00#
(PGUP)	4900	4939	8400	9900#
(PGDN)	5100	5133	7600	A100#
(↑)	4800	4838	8D00*	9800#
(↓)	5000	5032	9100*	A000#
(←)	4B00	4B34	7300	9B00#
(→)	4D00	4D36	7400	9D00#

	Normal	Shift	Ctrl	Alt
clavier numérique				
*	372A	!!!!	!!!!	3700#
-	4A2D	4A2D	8E00*	4A00#
+	4E2B	4E2B	9000*	4E00#
5	4C00*	4C35	8F00	9C00#

Notes:

---- Touches ignorées

* Touches en général non retournées par un clavier normal

\# Touches en général non retournées par un clavier étendu

!!!! Touches contrôlant l'impression d'écran et l'écho. Elle ne peuvent simuler cet effet avec un clavier étendu pour lequel les touches * et PtrSrc sont distinctes.

% Les touches (F11) et (F12) sont retournées seulement si elles exitent réellement sur le clavier

A quelques exceptions près, l'unité TPCLPLUS renvoie des résultats semblables à ceux du BIOS d'un clavier étendu. Une petite différence rend les claviers étendus plus simple à utiliser : les codes de balayage qui contiennent normalement $E0 dans l'octet de poids faible, pour indiquer que la touche est spécifique à un clavier étendu (telles que les touches isolées de déplacement curseur) contiendront 0 dans l'octet de poids faible.

Les touches marquées par un # sont normalement ignorées par le BIOS étendu. En définissant les touches du pavé numérique lorsque (ALT) est pressée, la possibilité habituelle d'entrer des touches est désactivée. Dans les programmes, la possibilité d'utiliser (ALT)-(SHIFT) suivi des flèches sera généralement plus utile. Néanmoins, vous pouvez encore saisir des touches sur le pavé numérique en pressant à la fois (ALT)-(SHIFT) (shift gauche) simultanément pendant la saisie du nombre. Exemple, pour saisir #196 vous presserez la touche (ALT) et la touche (SHIFT) (shift gauche) avec votre main gauche, saisirez 196 sur le pavé numérique avec votre main droite, puis libérerez (ALT) et (SHIFT) (shift gauche).

Remarquez que notre programme résident REF affiche tous les codes des touches spéciales renvoyés par TPCLPLUS. Si vous lancez le programme REF et observez la colonne intitulée 'Etendue', vous verrez que ces touches spéciales sont affichées en haute intensité ou blanc, ce qui vous permettra de distinguer facilement les codes standard des codes non standard.

Informations complémentaires

Le test utilisé par TPCLPLUS pour détecter la présence d'un clavier étendu est garanti avec les PS2, et à notre connaissance il n'a jamais indiqué qu'un clavier étendu était installé si tel n'était pas le cas. Néanmoins, il arrive parfois que ce test détecte mal la présence d'un clavier étendu. Pour remédier à ces situations, nous fournissons un petit programme appelé EKFIX.COM que vous pouvez utiliser dans vos propres programmes. Cet utilitaire positionne simplement à 1 le bit 4 de l'octet d'adresse $40:$96 si un clavier étendu est installé. Le message affiché par le programme indique si l'indicateur binaire est positionné à 1 (activé) ou à 0 (désactivé).

TPCLPLUS devrait être la première unité citée dans une clause Uses, notamment avec les programmes résidents. Si vous utilisez TPCLPLUS dans un programme qui utilise aussi TPTSR, TPCLPLUS doit précéder TPTSR dans la liste Uses pour que les vecteurs d'interruption soient restaurés dans l'ordre approprié.

Avertissement important : Comme cette unité utilise l'interruption $9, une erreur système peut se produire si *SideKick* est installé, et qu'un programme résident qui utilise l'interruption $9 a été chargé après *SideKick* ; (c'est pourquoi Borland recommande toujours de charger *SideKick* en dernier). Vous devrez prendre les mêmes précautions avec les programmes qui utilisent l'interruption $9 mais pour d'autres raisons. Cela ne concerne ni les programmes qui utilisent TPTSR, puisque cette unité désactive automatiquement le code de *SideKick* incompatible avec le gestionnaire d'interruptions $9, ni *SideKickPlus*.

TPCLPLUS : Déclarations

Constantes

```
ActiveEtendu : Boolean = True;
```

Additionner cette variable à "False" pour désactiver temporairement les gestionnaires d'interruption de TPCLPLUS.

Variables

```
ClavierEtendu : Boolean;
```

Cette variable est positionnée à "True" lorsqu'un clavier étendu a été détecté. Cette variable sert principalement à informer. La modification de cette valeur peut conduire à des résultats imprévisibles.

RestaureVecteursClavier

Déclaration

```
procedure RestaureVecteursClavier;
```

Objet

Restaure les vecteurs initiaux pour une interruption $09 et $16.

Commentaire

Cette routine restaure les vecteurs d'interruption qui ont été utilisés par TPCLPLUS au début d'un programme. Ces vecteurs sont restaurés automatiquement lorsque le programme se termine, mais dans certains cas il faudra les restaurer manuellement en appelant cette procédure. Le cas le plus fréquent est un programme résident qui vient d'appeler *DesactiveResid* pour se libérer lui-même.

Lorsque *RestaureVecteursClavier* a été appelé, il n'est plus possible de réactiver les gestionnaires d'interruption de l'unité. Si vous devez seulement désactiver les gestionnaires d'interruption temporairement, initialisez la variable globale *ActiveEtendu* à "False", puis à "True" lorsque vous désirez réactiver le gestionnaire d'interruption.

Exemple

```
uses TpClPlus, TpTsr;
...
if DesactiveResid then
   RestaureVecteursClavier
else
   {Affichage du message : pas prudent de décharger le TSR}
```

RestaureVecteursClavier est appelé dès que le programme résident doit être désactivé de la mémoire. Remarquez que TPCLPLUS apparaît avant TPTSR dans l'instruction Uses.

Voir aussi

DesactiveResid (TPTSR)

6 Interruptions et programmes résidents (TSR)

Généralités

L'écriture de routines utilisant les vecteurs d'interruption (ISR : Interrupt Routine Service) et l'écriture de programmes résidents fiables (TSR) sont certainement les tâches les plus redoutées des programmeurs sur PC. *Turbo Professionnel* vous propose un ensemble de routines qui vous feront gagner du temps.

Comme TPCRT, TPINT est l'une des unités les plus importantes de *Turbo Professionnel*. Elle sert principalement à gérer les vecteurs d'interruption. Lorsque vous appelez le DOS pour capturer un vecteur d'interruption, vous devez indiquer le numéro de l'interruption et l'adresse de votre routine. La fonction *InitVecteur* de TPINT vous demande en plus de spécifier un code d'identification (tel qu'un code d'identification de fichier DOS). Ce code d'identification vous permet de garder la trace du contexte et lorsque vous avez appelé *InitVecteur*, de ne plus vous soucier de votre routine de gestion des interruptions.

Si vous ne devez pas restaurer un vecteur d'interruption, appelez simplement *RestaureVecteur* avec le code d'identification, et TPINT renseignera le DOS avec le bon numéro d'interruption et l'adresse de l'ancienne routine. Que votre programme se termine normalement ou qu'il s'arrête sur une erreur d'exécution, vous n'avez pas à vous soucier du vecteur d'interruption que vous avez utilisé, puisque TPINT le restaurera automatiquement.

Turbo Pascal 4.0 apporte de gros avantages avec les procédures d'interruption qui sauvegardent et restaurent tous les registres CPU, initialisent le registre DS (Segment de Données), et gèrent une pile qui permet de traiter ces registres comme des paramètres. Cela vous permet d'écrire des routines de gestion d'interruption simples sans avoir recours au langage assembleur mais dans de nombreux cas, ce n'est pas suffisant. Aussi, le second objet de l'unité TPINT est de vous assister dans l'écriture de vos routines de gestion d'interruption spécialement si ces routines utilisent

beaucoup la pile. Elle contient des routines pour allouer et libérer des piles alternées, pour passer d'une pile à une autre depuis vos routines de gestion d'interruption, pour sortir rapidement d'un niveau d'interruption quel que soit le nombre de procédures appelées, se dérouter vers le gestionnaire d'interruptions précédent, et beaucoup d'autres possibilités.

```
{$S-,I-,R-}
{$M 1024,0,200}
program Bonjour;
uses
   Dos,TpInt,TpTsr;
const
   HotKey = $080F;          {code de Alt-Tab}
   ServicesDOS = True;      {Evite l'empilement au milieu du DOS}

{$F+}
procedure PointEntreePopup(var Regs : Registers);
begin
   writeln('Bonjour !');
end;
{$F-}

begin
   {définit la popup}
   if DefinitPop(HotKey,@PointEntreePopup,Ptr(SSeg,SPtr),ServicesDOS)
   then begin
      writeln('Popup en mémoire, tapez <Alt><Tab>
               pour l''activer.');
      {Rend les routines popup actives}
      PopupsOn;
      {essayons de rendre le programme résident}
      if not TermineEtResteResident(ParagraphesAConserver, 0) then {};
   end;

   {si nous arrivons ici, nous avons échoué}
   writeln('Impossible de rendre ce programme résident');
end.
```

Comme vous pouvez le constater, l'écriture d'un programme résident est très facile avec *Turbo Professionnel* et les résultats de vos efforts seront récompensés par des programmes fiables qui s'activeront lorsque vous le souhaitez, et non en plein milieu du DOS ; qui ne détruiront pas la pile des programmes interrompus ; qui s'activeront sur une combinaison de touches ; qui ne se chargeront pas deux fois (si vous tentez deux appels successifs) ; qui ne s'interrompront pas sur des erreurs critiques du DOS ; et qui pourront être facilement déchargés de la mémoire en toute sécurité ; (tous les programmes de démonstration de *Turbo Professionnel* utilisent cette facilité de s'exécuter et de rester résidents après leur exécution. Il existe même une possibilité pour faire communiquer des programmes résidents entre eux ou avec un gestionnaire d'erreur qui évite d'interrompre brutalement le système à la moindre erreur d'exécution.

L'unité TP8087 vous aidera à écrire des programmes résidents qui utilisent le coprocesseur mathématique (8087, 80287, et 80387) avec des nombres réels. La procédure *Sauve8087* vous permet de sauvegarder les registres du coprocesseur lorsque votre programme résident est activé et de les restaurer grâce à *Restaure8087* à la sortie. TP8087 offre également deux routines pour contrôler les erreurs. La procédure *Exceptions8087* vous permet d'activer ou de désactiver les interruptions exceptionnelles. La fonction *Erreur8087* renvoie l'erreur courante du coprocesseur, et vous permet de contrôler les erreurs de calculs. TP8087 supporte également un émulateur virgule flottante *Turbo Pascal 5.0*.

Un problème auquel les programmeurs de PC auront à faire face tôt ou tard, est la gestion des erreurs critiques du DOS, celles qui affichent le célèbre message "Abort, Retry, Ignore". Bien que les versions *Turbo Pascal* 4.0 et 5.0 gèrent ces erreurs, la méthode utilisée ne facilite pas l'interprétation de l'erreur. L'unité TPINT24 contient un gestionnaire d'erreurs critiques qui peut vous aider à distinguer clairement les erreurs critiques des autres erreurs. Remarquez que le répartiteur de routines popup de TPTSR installe automatiquement un gestionnaire d'erreurs critiques du DOS semblable à celui de TPINT24, avant d'appeler les routines popup qui utilisent les services du DOS.

TPINT : Routines de gestion d'interruption (ISR)

Comme nous l'avons vu au début de ce chapitre, l'unité TPINT a trois fonctions importantes, la principale étant les routines de gestion d'interruption (Interrupt Service Routine).

L'unité TPINT garde simplement la trace de vos codes d'identifications d'interruptions.

La procédure normale pour capturer un vecteur d'interruption en Turbo Pascal 4.0 est de (1) appeler la routine *GetIntVec* de l'unité DOS, (2) sauvegarder le vecteur courant, (3) appeler *SetIntVec* pour faire pointer le vecteur sur votre routine d'interruption et (4) restaurer l'ancien vecteur lorsque votre programme se termine. L'étape (4) est la plus difficile car un programme peut se terminer pour de nombreuses raisons et en de nombreux points. Dans la plupart des cas, TPINT vous permet de réduire ces quatre étapes en une seule :

```
const CodeIsr = 16;
------------------
If InitVecteur(NumInt, CodeIsr, @MonISR) then {};
```

TPINT sauvegardera le vecteur courant pour vous, le modifiera pour qu'il pointe sur *MonISR* et restaurera l'ancien vecteur lorsque votre programme se terminera.

TPINT peut garder la trace d'au plus 20 routines d'interruptions (*MaxISRs*), bien plus que n'importe quel programme peut vraisemblablement en traiter. Chacune d'entre elles devra être affectée à un code d'identification unique. Un code d'identification est un simple numéro utilisé pour repérer une routine particulière et dans la plupart des programmes vous pouvez utiliser n'importe quel numéro compris entre 1 et 20. N'oubliez pas cependant que deux autres unités de *Turbo Professionnel* qui utilisent TPINT réservent des codes d'identification particuliers pour leur propre utilisation. TPTSR se réserve déjà les numéros 1-11 et 13, et TMACRO utilise la 12ème. Puisque le nombre de codes d'identification réservés pour d'autres unités peut s'accroître dans le futur, nous vous recommandons d'utiliser seulement les codes d'identification n° 16 et au-delà, spécialement dans vos programmes résidents.

TPINT fournit également des routines qui simplifient l'écriture des routines d'interruption en Pascal. Toutes ces routines dépendent de deux choses : (1) "des procédures d'interruption" de Turbo Pascal et (2) des variables de types *RegistresInt* définies dans TPINT. Si vous avez étudié dans le manuel

Turbo Pascal les explications sur les procédures d'interruption, vous savez que vous pouvez traiter les registres CPU comme des paramètres en déclarant la procédure d'interruption comme :

```
procedure CdIsr(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP : Word);
   interrupt;
begin
end;
```

Afin d'utiliser les procédures dont nous venons de parler, nous modifierons simplement cette déclaration en :

```
procedure MonISR(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
begin
end;
```

Tout ce que vous devez faire maintenant est d'ajouter la déclaration de *Regs*. Vous en tirerez deux avantages. Premièrement, vous pouvez maintenant référencer les demi-registres (AH, AL, BH, Bl, etc) en utilisant "Regs.AH", "Regs.AL" et ainsi de suite. Deuxièmement, vous pouvez maintenant traiter les différents registres en groupe et les passer tous ensembles comme un seul paramètre à une procédure.

Maintenant, si vous avez écrit votre routine d'interruption comme nous l'avons indiqué, vous pouvez utiliser les cinq routines qui traitent de ce sujet. *RetourInt* permet de revenir d'une interruption à partir d'une procédure autre que votre routine d'interruption. Par exemple, vous pouvez appeler une routine telle que celle-ci :

```
procedure ProgInt(var Regs : RegistresInt);
begin
   {--- traitement ---}
   RetourInt(Regs);
end;
```

ChainInt permet de terminer votre interruption et de passer le contrôle à l'ancien vecteur d'interruption tel que :

```
ChainInt(Regs, Tableau_ISR[CodeIsr].AdrOrig);
```

EchPileEtAppelNear et *EchPileEtAppel* permettent d'activer une autre pile et appellent une autre procédure à partir de votre routine d'interruption. Lorsque la procédure que vous avez appelée rend le contrôle, l'ancienne pile est restaurée.

EmuleInt permet d'appeler l'ancien vecteur d'interruption depuis votre routine d'interruption sans sortir. Par exemple :

```
EmuleInt(Regs, Tableau_ISR[CodeIsr].AdrOrig);
```

Enfin, TPINT fournit deux fonctions qui permettent d'allouer ou de libérer facilement des piles alternées. *AllouePile* alloue sur le tas une pile (en réalité, un tableau) d'une taille spécifiée et renvoie un pointeur sur la fin du tableau puisque la pile croît vers le haut. La procédure conjointe *DeallouePile* libère la pile allouée de cette façon.

Rappel : Ainsi que le mentionne le manuel de *Turbo Pascal* (partie "Au coeur de *Turbo Pascal*", sous le titre "Ecriture des procédures d'interruptions"), les routines d'entrée/sortie et de gestion de mémoire de *Turbo Pascal* ne sont pas réentrantes et ne peuvent pas être utilisées dans des routines de gestions d'interruptions. Le DOS n'est pas réentrant, aussi les services du DOS ne peuvent être utilisés et, finalement, de nombreuses routines de *Turbo Professionnel* ne sont pas réentrantes. Pour en être sûr, contrôlez le code source de la routine : toute routine qui modifie une variable globale déclarée dans une unité *Turbo Professionnel* est probablement non réentrante.

Constantes

```
MaxISRs = 20;
```

Nombre maximum de routines de gestion d'interruptions qui peuvent être gérées par TPINT.

Types

```
Dummy5 = array[1..5] of Word;
```

Utilisé pour compléter les déclarations de *RegistresInt*.

```
RegistresInt = record
   case byte of
      1 : (BP, ES, DS, DI, SI, DX, CX, BX, AX, IP, CS, Flags: Word);
      2 : (Dummy : Dummy5; DL, DH, CL, CH, BL, BH, AL, AH : Byte);
   end;
```

Enregistrement qui est le reflet du contenu de la pile dans une procédure d'interruption.

```
Enreg_ISR = record
   NumInt : Byte;
   AdrOrig : Pointer;
   NouvAdr : Pointer;
   Capture : Boolean;
end;
```

Enregistrement contenant les données sur une routine de gestion d'interruption donnée. *NumInt* est le vecteur qui a été activé, *AdrOrig* est l'adresse de la routine d'interruption précédente, *NouvAdr* est l'adresse de la routine d'interruption actuelle et *Capture* est utilisé en interne pour déterminer si le vecteur a été déjà restauré ou non.

Variables

```
Tableau_ISR : array[1..MaxISRs] of Enreg_ISR;
```

Tableau global des enregistrements contenant les données sur les routines d'interruption gérées par TPINT.

AllouePile

Déclaration

```
function AllouePile(var P : Pointer; TailleEnOctets : Word) : Boolean;
```

Objet

Alloue une pile d'une taille égale à TailleEnOctets.

Commentaires

Cette fonction fournit un moyen pratique d'allouer des piles pour des routines d'interruption et des "popups". *TailleEnOctets* est la taille de la pile allouée et ne doit pas dépasser 65515. Si une pile d'une taille désirée peut être allouée, *AllouePile* renvoie un pointeur vers la fin de la pile dans *P*. Sinon, la fonction renvoie "False".

Très important : N'utilisez pas *FreeMem* pour libérer une pile allouée avec *AllouePile*, utilisez au contraire *DeallouePile*.

Exemple

```
if not AllouePile(MaPile, TailleDeMaPile) then begin
   Writeln('Mémoire insuffisante. Abandon...');
   Halt(1);
end;
```

AllouePile est appelée pour allouer une pile de *TaillePileOctets*. S'il n'y a pas assez de place sur le tas, la fonction renvoie "False" et le programme s'interrompt.

Voir aussi

DeallouePile (TPINT)

ChainInt

Déclaration

```
procedure ChainInt(var Regs : RegistresInt; AdrSaut : Pointer);
```

Objet

Restaure la pile des registres et saute à *AdrSaut*.

Commentaires

Cette macro inline permet de restaurer les registres CPU empilés au début de la procédure d'interruption puis passe le contrôle à la routine d'interruption qui fournit le service demandé au vecteur que vous avez activé. Contrairement à la macro *SautAncIsr* de TPINLINE.PAS, *ChainInt* est destinée tout d'abord aux routines d'interruption écrites en Pascal plutôt qu'en code inline, et peut être appelée à partir d'une procédure différente de la procédure réelle d'interruption. (A condition que *Regs* ait été passé comme paramètre par adresse à la seconde procédure).

Exemples

```
if not InitVecteur($09, MonGestISR, @MonISR) then
   {Erreur};
...
procedure MonISR(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
begin
{..Suite..}
   ChainInt(Regs, Tableau_ISR[MonGestISR].AdrOrig^);
end;
```

Incorrect ! Le '^' dans *'Tableau_ISR[MonGestISR].AdrOrig^'* n'est pas nécessaire.

```
   ChainInt(Regs, Tableau_ISR[MonGestISR].AdrOrig);
```

ChainInt restaurera les registres et sautera à la routine d'interruption précédente.

Voir aussi

EmuleInt (TPINT)

SautAncIsr (TPINLINE)

DeallouePile

Déclaration

```
procedure DeallouePile(P : Pointer);
```

Objet

Libère une pile qui a été allouée.

Commentaires

Cette routine permet de libérer facilement une pile allouée avec *AllouePile*. Remarquez que la taille de la pile n'est pas demandée en paramètre puisque *AllouePile* aura déjà enregistré cette information (juste au-dessus du sommet de la pile) pour que *DeallouePile* puisse l'utiliser.

Très important : n'utilisez pas *DeallouePile* pour libérer une pile qui n'a pas été allouée avec *AllouePile*. Utilisez *Dispose* ou *FreeMem* dans ce cas.

Exemple

```
if not AllouePile(MaPile, TailleDeMaPile) then
   {Erreur};
...
DeallouePile(MaPile);
```

DeallouePile est utilisée pour libérer une pile allouée avec *AllouePile*.

Voir aussi

AllouePile (TPINT)

EchPileEtAppel

Déclaration

```
procedure EchPileEtAppel(Routine, PP : Pointer;
                         var Regs : registresInt);
```

Objet

Active la pile désignée par *PP* et appelle la routine avec *Regs* en paramètre.

Commentaires

Cette routine permet de changer de pile (pointée par *PP*) et d'appeler une procédure en passant *Regs* en paramètre. La routine appelée doit être de la forme :

```
{$F+}
procedure Proc(var Regs : RegistresInt);
begin
{-- traitement --}
end;
{$F-}
```

Très important : *Proc* doit être appelée avec l'option FAR à partir de la routine d'interruption actuelle. Sinon, vous devrez utiliser *EchPileEtAppelNear* à la place.

Exemple

```
{$F+}
procedure NonISR(var Regs : RegistresInt);
begin
   {..Suite..}
   if RetourNecessaire then
      RetourInt(Regs);
   {..Suite..}
end;
{$F-}

procedure MonISR(BP : Word); interrupt;
var Regs : RegistresInt absolute BP;
begin
   if Regs.AX = ValeurMagique then
      EchPileEtAppel(@NonIsr, MonPtrPile^, Regs)
   else begin
      {..Suite..}
   end;
end;
```

Incorrect ! Le '^' dans *'MonPtrPile^'* n'est pas nécessaire.

```
EchPileEtAppel(@NonIsr, MonPtrPile, Regs)
```

EchPileEtAppel activera la pile pointée par *MonPtrPile* et appelera *NonIsr* en passant *Regs* en paramètre. Remarquez que *NonIsr* a été compilée avec l'option FAR.

Voir aussi

EchPileEtAppelNear (TPINT)

EchPileEtAppelNear

Déclaration

```
procedure EchPileEtAppelNear(Routine : Word; PP : Pointer;
                             var Regs : RegistresInt);
```

Objet

Active la pile désignée par *PP* et appelle *Routine* avec *Regs* en paramètre.

Commentaires

Comme *EchPileEtAppel,* cette routine permet de changer de pile (pointée par *PP*) et d'appeler une procédure en passant *Regs* comme paramètre. La routine appelée doit être de la forme

```
procedure Proc(var Regs : RegistresInt);
   begin
      {-- traitement --}
   end;
```

Très important : *Proc* doit être appelée avec l'option NEAR à partir de la routine d'interruption actuelle. Sinon, vous devrez utiliser *EchPileEtAppel* à la place.

Exemple

```
procedure NonISR(var Regs : RegistresInt);
begin
   {..Suite..}
   if RetourNecessaire then RetourInt(Regs);
   {..Suite..}
end;

procedure MonISR(BP : Word); interrupt;
var Regs : RegistresInt absolute BP;
begin
   if Regs.AX = ValeurMagique then
      EchPileEtAppelNear(Ofs(NonIsr), MonPtrPile^, Regs)
   else begin
      {..Suite..}
   end;
end;
```

Incorrect ! Le '^' dans *'MonPtrPile^'* n'est pas nécessaire.

```
EchPileEtAppelNear(Ofs(NonIsr), MonPtrPile, Regs)
```

EchPileEtAppel activera la pile pointée par *MonPtrPile* et appelera *NonIsr* en passant *Regs* en paramètre. Remarquez que *NonIsr* n'a pas été compilée avec l'option FAR.

Voir aussi

EchPileEtAppel (TPINT)

EmuleInt

Déclaration

```
procedure EmuleInt(var Regs : RegistresInt; AdrInt : Pointer);
```

Objet

Emule une interruption.

Commentaires

Cette routine simule une interruption réelle en renseignant les registres CPU avec des valeurs contenues dans *Regs*, efface les interruptions, empile les indicateurs et effectue un appel FAR à *AdrInt*. Comme *AppelAncIsr* de TPINLINE.PAS, elle permet d'appeler une routine d'interruption gérant un vecteur d'interruption particulier avant que vous l'ayez activé. Contrairement à *AppelAncIsr*, *EmuleInt* est destinée à l'origine aux routines d'interruptions écrites en Pascal plutôt qu'aux routines en code inline, et elle peut être appelée à partir d'une procédure différente de la procédure d'interruption réelle (à condition que *Regs* soit passé en paramètre par adresse à la deuxième procédure).

Une deuxième utilisation courante de *EmuleInt* est d'appeler les routines via un pointeur. Voir les explications de l'unité TPTSR pour plus de détails.

Exemple

```
if not InitVecteur($09, MonGestISR, @MonISR) then
   {Erreur};
...
procedure MonISR(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
begin
   {..Suite..}
   EmuleInt(Regs, Tableau_ISR[MonGestISR].AdrOrig^);
end;
```

Incorrect ! Le '^' dans *'Tableau_ISR[MonGestISR].AdrOrig^'* n'est pas nécessaire.

```
   EmuleInt(Regs, Tableau_ISR[MonGestISR].AdrOrig);
```

EmuleInt rechargera les registres CPU avec le contenu de *Regs*, appelera la routine d'interruption précédente et chargera *Regs* avec le nouveau contenu des registres CPU.

Voir aussi

AppelAncIsr (TPINLINE) ChainInt (TPINT)

EnvoiFinInt

Déclaration

```
procedure EnvoiFinInt;
```

Objet

Envoie un signal de fin d'interruption au contrôleur d'interruption programmé.

Commentaire

Cette macro inline envoie un signal de fin d'interruption (End Of Interrupt) au contrôleur d'interruption programmé (Programmable Interrupt Controller). Elle ne doit être utilisée qu'avec des gestionnaires d'interruption relatifs aux interruptions matérielles telles que $8 (top horloge) ou $9 (clavier).

Exemples

```
procedure Int9(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
   B : Byte;
   AncInt9 : pointer;
begin
   {Lecture du code de balayage de pression/relâchement de touche}
   B := Port[$60];
   ...
   if not BesoinLireTouche then
      ChainInt(Regs, AncInt9)
   else begin
      {Réinitialise le clavier}
      B := Port[$61];
      Port[$61] := B or $80;
      Port[$61] := B;
      {Signal de fin d'interruption}
      EnvoiFinInt;
   end;
end;
```

Un bon exemple d'utilisation de *EnvoiFinInt* : un gestionnaire de l'interruption $9 qui lit les touches du clavier sous certaines conditions.

InitVecteur

Déclaration

```
function InitVecteur(NumInt, Identificateur : Byte;
                     RoutineUtl : Pointer) : Boolean;
```

Objet

Initialise une routine de gestion d'interruption.

Commentaires

La routine principale de l'unité TPINT, *InitVecteur*, permet de modifier un vecteur d'interruption (*NumInt*) pour pointer sur une routine de gestion d'interruption que vous avez écrite. (Pointée par *RoutineUtl*). Le paramètre *Identificateur* est le numéro que vous voulez utiliser pour identifier la routine d'interruption en question (voir les généralités au début de cette partie pour des explications sur les codes d'identification). Lorsque vous avez appelé *InitVecteur*, vous devez simplement traiter cette interruption. Lorsque votre programme se termine, l'ancien vecteur sera restauré automatiquement même si une erreur d'exécution se produit. Si vous devez restaurer le vecteur avant que le programme se termine, vous pouvez le faire avec *RestaureVecteur* (ou *RestaureTousVecteurs*). *InitVecteur* retourne "False" seulement si le code d'identification spécifié est supérieur à *MaxISRs* ou s'il est déjà utilisé.

Exemples

```
if not InitVecteur($09, MonGestISR, @MonISR) then
   {Le plus souvent, Gestionnaire déjà utilisé};
```

Ici *InitVecteur* est utilisée pour faire pointer le vecteur d'interruption $09 sur *MonIsr*.

```
RestaureVecteur(MonGestISR);
Code := ExecDos('', true, Nil);
if not InitVecteur($09, MonGestISR, @MonISR) then
   {Le cas ne doit pas se présenter};
```

Dans cet exemple, *RestaureVecteur* est appelée pour restaurer le code d'identification de l'ancienne interruption $09 avant de retourner sous le DOS et *InitVecteur* est ensuite appelée pour récupérer le vecteur.

Voir aussi

RestaureTousVecteurs (TPINT) RestaureVecteur (TPINT)

InterruptsOff

Déclaration

```
procedure InterruptsOff;
```

Objet

Désactive les interruptions.

Commentaires

Cette macro inline désactive simplement les interruptions.

Voir aussi

InterruptsOn (TPINT)

InterruptsOn

Déclaration

```
procedure InterruptsOn;
```

Objet

Active les interruptions.

Commentaires

Cette macro inline active simplement les interruptions.

Exemple

```
procedure MonISR(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
begin
   InterruptsOff;
   {..Suite..}
   InterruptsOn;
end;
```

Les interruptions sont désactivées au début de la routine d'interruption et réactivées à nouveau à la fin.

Voir aussi

InterruptsOff (TPINT)

RestaureTousVecteurs

Déclaration

```
procedure RestaureTousVecteurs;
```

Objet

Restaure tous les vecteurs d'interruption capturés.

Commentaires

Cette routine permet de restaurer tous les vecteurs d'interruption que vous avez capturés. Elle est destinée à l'origine aux usages internes puisque *RestaureTousVecteurs* est appelé automatiquement lorsque votre programme se termine.

Exemple

```
if not InitVecteur($09, MonGestISR, @MonISR) then {};
if not InitVecteur($09, MonAutreGestISR, @MonAutreISR) then {};
...
RestaureTousVecteurs;
```

Voir aussi

RestaureVecteur (TPINT)

RestaureVecteur

Déclaration

```
procedure RestaureVecteur(Identificateur : Byte);
```

Objet

Restaure un vecteur d'interruption.

Commentaires

Cette routine permet de restaurer un vecteur d'interruption en appelant la routine *InitVecteur*. Puisque tout vecteur d'interruption capturé est restauré automatiquement lorsque votre programme se termine, l'utilisation principale de *RestaureVecteur* est de restaurer un vecteur temporairement. Par exemple, CALC.PAS appelle *InitVecteur* chaque fois que la routine "popup" est activée pour détourner temporairement l'interruption $0 et gérer les erreurs arithmétiques, puis appelle *RestaureVecteur* en sortant de la routine "popup " pour restaurer l'ancien vecteur.

Exemple

```
if not InitVecteur($09, MonGestISR, @MonISR) then
   {Gestionnaire dèjà utilisé ?};
...
RestaureVecteur(MonGestISR);
Code := ExecDos('', True, Nil);
if not InitVecteur($09, MonGestISR, @MonISR) then {};
```

Restaure le code d'identification de l'ancienne interruption $09 avant de provoquer un accès au DOS.

Voir aussi

Initvecteur (TPINT)

RestaureTousVecteurs (TPINT)

RetourInt

Déclaration

```
procedure RetourInt(var RegsInt : RegistresInt);
```

Objet

Retour d'interruption.

Commentaires

Cette routine permet de sortir d'un niveau d'interruption quel que soit votre programme à condition que la procédure actuelle ait passée en paramètre par adresse de type *RegistresInt* lors de l'appel de la routine d'interruption. *RetourInt* est nécessaire seulement lorsque la pile a été modifiée, vraisemblablement en utilisant *EchPileEtAppel* ou *EchPileEtAppelNear* ou lorsque vous avez besoin de sortir d'une interruption depuis une procédure appelée par la routine d'interruption actuelle.

Exemple

```
procedure NonISR(var Regs : RegistresInt);
begin
   {..Suite..}
   if RetourNecessaire then
      RetourInt(Regs);
   {..Suite..}
end;

procedure MonISR(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
begin
   if Regs.AX = ValeurMagique then
      NonIsr(Regs)
   else begin
      {..Suite..}
   end;
end;
```

Dans cet exemple, *MonISR* appelle *NonIsr* en passant *Regs* comme paramètre et *NonIsr* utilise *RetourInt* pour sortir de l'interruption.

Voir aussi

EchPileEtAppel (TPINT) EchPileEtAppelNear (TPINT)

TPTSR : Gestion des programmes résidents

Note importante : L'unité TPRESID de *Turbo Professionnel 4.0* a été renommée en TPTSR afin de tenir compte des terminologies couramment employées.

L'objet principal de l'unité TPTSR est de fournir un système de gestion des programmes résidents facile à utiliser et fiable. En suivant nos conseils, vous pourrez écrire des programmes résidents fiables aussi facilement que des programmes non résidents. Mais avant d'expliquer comment le faire, définissons quelques termes.

Un programme résident (en anglais, Terminate and Stay Resident) est un programme qui, tout de suite après avoir été chargé, redonne le contrôle au système d'exploitation (DOS) tout en restant en mémoire. Chaque programme résident contient une ou plusieurs routines "popup", appelées par le *répartiteur de tâches* lorsqu'un événement particulier se produit. Le travail du répartiteur de tâches peut être divisé en six parties : (1) contrôler le système pour surveiller les événements qui l'intéressent ; (2) lorsqu'un tel événement se produit, définir quelle "popup" doit être appelée ; (3) en fonction de ce que l'on sait sur la "popup", déterminer si elle peut être ou non activée en toute sécurité ; (4) avant d'appeler la "popup", sauvegarder l'état du système et prendre toutes les précautions pour assurer la sécurité de la "popup" ; (5) appeller la "popup" ; (6) restaurer l'état du système. (*Remarque* : L'ordre des événements décrits ici et dans les paragraphes suivants, a été décomposé intentionnellement afin de rendre ce processus complexe plus facile à comprendre).

Maintenant, regardons l'exemple simple "Bonjour à tous !" montré au début de ce chapitre pour voir ces principes en action. Le répartiteur de tâches contrôle le clavier, attend la frappe de la touche (TAB) en même temps que la touche (ALT). Lorsque cela se produit, le répartiteur contrôle ses tables et s'aperçoit que la procédure *PointEntreePopup* doit être appelée et qu'il a besoin d'accéder aux services du DOS (*ServicesDOS* est à "True"). Le répartiteur vérifie alors que tous ces services peuvent être utilisés en toute sécurité et en considérant qu'ils le peuvent, crée un environnement stable pour la "popup". Finalement, *PointEntreePopup* est appelée. Une fois terminée, elle rend le contrôle au répartiteur qui restaure alors l'état du système, rend le CPU à la tâche qui a été suspendue lorsque (ALT)-(TAB) a été pressé et redonne le contrôle au système.

Nombre et types de "popups"

Par défaut, TPTSR vous permet de définir jusqu'à 8 (*MaxPopups*) routines "popup" dans un programme, chacune d'elles appartenant à l'une des deux catégories. La catégorie la plus utilisée est associée à une combinaison de touches (hotkey), l'autre catégorie de "popup" n'est pas activée sur un événement donné. Vous pouvez l'invoquer quand vous le désirez en appelant la procédure *InitHorlogePop* avec le code d'identification affecté à la "popup" (voir ci-dessous) et un compteur d'horloge (*TempsLimite*). Cela indique au répartiteur de commencer sa surveillance et d'attendre une bonne occasion pour appeler la "popup". Si le compteur d'horloge dépasse *TempsLimite* sans qu'une telle occasion se soit présentée, la "popup" ne sera pas appelée.

Combinaisons de touches (Hotkeys)

La "popup" la plus courante est celle qui est invoquée lorsque des touches spécifiques sont pressées simultanément, une combinaison désignée par le terme "hotkey". Pour *Turbo Professionnel*, une "hotkey" est composée d'une touche en combinaison avec des touches shift. Outre les touches shift normales, (SHIFT) (gauche) et (SHIFT)(droite), les touches (ALT) et (CTRL) sont également traitées comme des touches shift. Pour spécifier la hotkey de votre "popup", vous devrez définir une constante de type Word dont les bits ont les valeurs suivantes :

Bit	Valeur	Signification
15-12	$0000	inutilisée, doit être à 0
11	$0800	Alt
10	$0400	Ctrl
9	$0200	shift gauche
8	$0100	shift droite
7-0	$00xx	Code de balayage des touches principales

Par exemple, supposons que vous vouliez que votre "popup" soit activée par (ALT)-(SHIFT) (gauche) et **Z**. Le code de balayage pour **Z** est $2D (voir le tableau des codes de balayage en appendice E du manuel de *Turbo Pascal 4.0*, et en appendice C du manuel de *Turbo Pascal 5.0*) ce qui nous donne la valeur pour l'octet de poids faible, bits 0-7. Si nous ajoutons à cela les valeurs pour (ALT)-(SHIFT) (gauche) , nous obtenons $0800 + $0200 + $002D = $0A2D. (Le programme CALC est très pratique pour faire des calculs hexadécimaux tels que ceux-ci).

Compléments sur les "hotkeys"

Vous devez garder à l'esprit deux autres points concernant les "hotkeys". Premièrement, si cela n'était pas évident dans le paragraphe précédent, il n'est pas possible de définir une "popup" qui soit activée par une seule touche. L'une des quatre touches shifts doit être spécifiée. Si l'octet de poids fort de la "hotkey" est à 0, la "popup" ne sera jamais appelée. Il est possible, néanmoins, de traiter une des touches shift comme une touche principale. Par exemple, pour activer une "popup" sur (CTRL)-(SHIFT) (gauche) , vous pourrez spécifier soit $042A ou $021D, en fonction de la touche qui sera appuyée la première. Deuxièmement, vous êtes limité à une "hotkey" par "popup". Bien que *DefinitPop* vous permette de définir deux fois la même "popup" en utilisant des "hotkeys" différentes, cela est potentiellement dangereux et il vaut mieux l'éviter.

Codes d'identification de "popup"

Lorsque vous appelez *DefinitProcPop* pour définir une "popup" qui est appelée comme résultat d'un appel à *InitHorlogePop*, vous devez spécifier un code d'identification. Contrairement aux unités TPECRAN et TPINT qui vous permettent d'assigner vos propres codes d'identification, TPTSR les assigne à votre place. Aussi, tout ce que vous devrez faire est de déclarer une variable globale de type byte -- disons "*MonCodeIdentPopup*" -- et la passer comme paramètre. Si certains codes d'identification sont encore disponibles, *DefinitProcPop* assignera une valeur à *MonCodeIdentPopup* et renverra "True". Lorsque vous serez prêt pour appeler votre "popup", vous devrez écrire quelque chose comme :

```
InitHorlogePop(MonCodeIdentPopup, 18);
```

qui signifie "pendant les 18 prochains tops d'horloge, attendez l'occasion d'appeler la "popup" associée avec *MonCodeIdentPopup*. Les "popups" activées par les hotkeys ont aussi des codes d'identifications mais vous n'avez pas besoin de les connaître puisqu'ils sont réservés aux utilisations internes seulement.

Routines "popup"

Toutes les "popups" écrites avec *Turbo Professionnel* qui sont activées en appuyant sur une touche ou en appelant *InitHorlogePop* doivent être de la forme :

```
{$F+}
procedure SquelettePopup(Regs : Registers);
begin
   {-- corps de la popup--}
end;
{$F-}
```

La directive {$F+} et la déclaration du paramètre *Registers* sont importantes. Si l'une ou l'autre manque, votre "popup" sera appelée correctement mais se terminera anormalement. Lors de l'appel à *DefinitPop/DefinitProcPop*, le paramètre "*Routine*" doit être un pointeur sur la "popup" tel que : "*@SquelettePopup*".

Deux autres points moins importants. Premièrement, le paramètre *Regs* doit être du type *Registers* (défini dans l'unité DOS) et *non* du type *RegistresInt* (défini dans TPINT), si vous désirez *référencer* les valeurs des registres. Deuxièmement, si vous voulez *modifier* des valeurs des registres, vous devrez passer le paramètre *Regs* par adresse. La modification des valeurs n'est pas recommandée mais techniquement possible.

Piles

Chaque "popup" doit avoir sa propre pile. Cela signifie que le paramètre *PtrPile* passé aux procédures *DefinitPop/DefinitProcPop* doit être différent pour chaque "popup" dans un programme. La plupart des programmes résidents sont composés d'une seule popup et dans ce cas, vous pouvez toujours utiliser "*Ptr(SSeg, SPtr)*" comme paramètre. Mais, les programmes qui contiennent plusieurs "popups" (tel que SMACS.PAS et GESTEMPS.PAS) doivent allouer de la place pour plusieurs piles en déclarant un tableau comme variable globale ou en appelant la routine *AllouePile* de TPINT qui alloue un tableau sur le tas. Dans les deux cas, le paramètre *PtrPile* doit pointer sur le dernier élément du tableau plutôt que sur le premier. (Remarquez que la routine *AllouePile* renvoie toujours un pointeur sur la fin du tableau).

Demande d'accès au DOS

La plupart des routines "popup" ont besoin d'accéder aux services du DOS, mais pas toutes. Dans le doute, considérez qu'elles le doivent. Mais dans le cas contraire (par exemple, si vous devez afficher un texte sur l'écran avec la routine *EcritVite* de TPCRT), vous devrez passer "False" à la routine *DefinitPop* dans le paramètre *ServicesDos*. Ce qui vous permettra de lancer une routine "popup" dans des situations où vous ne le pourriez pas.

Les "services du DOS" couvrent de larges besoins. Toutes les entrées/sorties sur disque, ainsi que de nombreuses fonctions qui pourraient vous surprendre, font appel aux services du DOS. Les routines *GetDate* et *GetTime* de l'unité DOS, par exemple, utilisent les services du DOS. Surtout, si avez des doutes, considérez que les services du DOS sont nécessaires.

Précautions à prendre avant de lancer un programme résident

Avant de lancer un programme résident, surtout si vous utilisez des unités TPCRT ou CRT dans votre programme, et que vous utilisez *Turbo Pascal* 4.0, une précaution supplémentaire est nécessaire.

Si vous consultez le code source de n'importe quel programme de démonstration résident en mémoire, vous trouverez toujours l'instruction suivante vers la fin :

```
{IFDEF Ver40}
SetIntVec($1B, SaveInt1B);
{$ENDIF}
```

Cette instruction restaure le vecteur pour l'interruption $1B générée par le BIOS lorsque (CTRL)-(BREAK) est pressé ; ce vecteur est capturé à la fois par CRT et TPCRT. Ce vecteur doit être restauré avant de lancer un programme résident. Cela est automatique avec *Turbo Pascal* 5.0 mais pas avec *Turbo Pascal* 4.0. De la même façon, si une partie non résidente de votre programme capture n'importe quel autre vecteur d'interruption non utilisé par la partie résidente, ce vecteur devra être restauré avant de lancer le programme résident.

Lancement d'un programme résident

Après avoir défini vos "popups", que devez-vous faire ? Dans la plupart des cas, vous devrez simplement appeler la fonction *TermineEtResteResident* de cette façon :

```
if not TermineEtResteResident(ParagraphesAConserver, 0) then {erreur};
```

ParagraphesAConserver détermine la taille de la mémoire que votre programme utilise réellement. L'instruction ci-dessus signifiera à *TermineEtResteResident* de rendre au DOS tout ce que vous n'utilisez pas. Généralement, c'est préférable, mais si vos "popups" doivent allouer ultérieurement de la place sur le tas, elles n'auront plus de mémoire disponible. Ainsi, en supposant que vous vouliez réserver 4000 octets (250 paragraphes) sur le tas pour vos "popups", vous devrez écrire à la place :

```
if not TermineEtResteResident(ParagraphesAConserver+250, 0)
   then {erreur};
```

Note importante : il n'est pas absolument nécessaire de donner une valeur particulièrement petite pour la taille maximum sur le tas d'un programme résident (en utilisant la directive de compilation {$M}). Au contraire, si vous devez allouer quelque chose sur le tas après avoir lancé un programme résident, c'est une bonne idée d'utiliser une valeur plus importante pour que *TermineEtResteResident* place le tas vers le bas. Le paramètre *ParagraphesAConserver* passé à *TermineEtResteResident* ne doit pas demander plus d'espace sur le tas qu'il y en a de disponible.

Exécution de votre programme résident

Au cas où vous l'auriez oublié, un programme résident doit être chargé à partir d'une commande DOS. Si vous êtes habitué à lancer vos programmes à partir du compilateur Turbo, vous devrez modifier vos habitudes de

programmation. De la même façon, la plupart des debuggers ne peuvent être utilisés pour mettre au point des programmes résidents, y compris *TDebugPLUS* 4.0, le *Turbo Debugger* de Borland et le debugger intégré de *Turbo Pascal* 5.0.

Erreurs d'entrée/sortie et erreurs critiques du DOS

Le répartiteur avant d'appeler une "popup" qui nécessite des accès aux services du DOS, doit installer un *gestionnaire d'erreurs critiques du DOS* (tel que TPINT24) pour traiter les erreurs qui entraînent l'affichage du message "Abort, Retry, Ignore". Ces messages sont désagréables mais ils peuvent devenir désastreux pour un programme résident. Le gestionnaire d'erreurs installé par le répartiteur demande simplement au DOS d'ignorer l'erreur (sous DOS 2.x) ou de refuser l'appel (sous DOS 3.0 ou plus). Ceci implique que vous reviendrez dans votre programme mais la condition d'erreur existera toujours, vous devez donc prévoir un traitement pour cette erreur. Comment ? En tête de votre programme, ajoutez la directive {$I-} pour que votre programme ne soit pas interrompu à l'exécution lorsqu'une erreur d'entrée/sortie se produit. Après chaque opération d'entrée/sortie qui pourrait vraisemblablement entraîner une erreur, ajoutez un appel à *ResultInt24*. Si elle renvoie autre chose que 0, une erreur d'entrée/sortie ou une erreur critique du DOS s'est produite et vous devrez la traiter comme si vous aviez appelé *IoResult* à la place.

Erreurs d'exécution

Les erreurs d'exécution peuvent être également fatales. TPTSR contient un gestionnaire d'erreurs qui essaie d'assister le système lorsque des erreurs se produisent à l'exécution mais ce n'est pas toujours possible. Heureusement, cela est généralement possible si votre "popup" utilise une pile alternée (sur le segment de données ou sur le tas). Aussi, il vous est conseillé d'utiliser l'une ou l'autre lors du développement de votre programme (un programme résident au point ne devrait pas générer d'erreur d'exécution, sinon cela signifie qu'il contient des erreurs). Remarquez que ce gestionnaire d'erreurs n'affiche pas de message et s'il réussit son opération d'assistance, le gestionnaire d'erreurs de Turbo qui affiche des messages ne sera jamais activé. Si vous voulez connaître l'erreur qui s'est produite, vous devrez écrire votre propre gestionnaire d'erreurs en suivant les étapes décrites dans le manuel de *Turbo Pascal.*

Utilisation de TPTSR avec TPFENETR ou TPMENU

Lorsque vous utiliserez TPFENETR ou TPMENU dans des programmes écrits avec TPTSR, souvenez-vous que TPFENETR et TPMENU utilisent librement le tas. La mémoire est réservée lorsque des fenêtres et des menus sont définis mais ces unités allouent / libèrent aussi des petits blocs de mémoire lorsque des menus et des fenêtres sont affichés / effacés de l'écran.

En général, cela signifie que vous devrez 1) Appelez *InitFen*, *SousMenu* et *ElemMenu* avant d'installer le programme résidant, et 2) garder au moins quelques centaines d'octets sur le tas au moment d'installer le programme résident. Par exemple,

```
if not TermineEtResteResident(ParagraphesAConserver+25, 0)
   then {erreur};
```

Réserve au moins 400 (25*16) octets sur le tas : lorsque le programme résident est installé. Voir "Installation de programme résident" pour des explications sur l'allocation du tas pendant l'installation d'un programme résident.

Naturellement, ces conseils s'appliquent à *toute unité ou routine* qui doit allouer de la place sur le tas, tel que TPTRI.

Utilisation de la pile et du tas

Une tâche difficile que TPTSR ne peut faire pour vous est de déterminer la taille de la pile ou du tas pour une "popup" particulière. Notre conseil est le suivant : pendant les étapes de développement, allouez plus d'espace sur la pile et le tas et ne vous souciez pas de l'utilisation de la mémoire. Lorsque le programme est pratiquement terminé, utilisez l'unité TPSTACK qui se trouve dans le sous répertoire BONUS de la disquette SRC3, pour effectuer des mesures précises du tas et de la pile. TPSTACK est destinée principalement à des programmes non résidents. Voir la documentation de TPSTACK (dans le sous répertoire BONUS) pour plus de détails.

Compléments sur la programmation des programmes résidents

Il y a certaines petites tâches que TPTSR ne peut pas gérer à votre place. Si vous écrivez un programme résident qui utilise des réels 8087, vous devrez sauvegarder et restaurer l'état du coprocesseur mathématique dans votre programme. L'unité TP8087 offre les routines nécessaires : *Sauve8087* et *Restaure8087*. Le gestionnaire de programme résident vous en laisse la responsabilité mais garantit cependant que les vecteurs d'interruption utilisés par l'émulateur virgule flottante seront correctement initialisés au lancement de votre programme résident.

De la même façon, l'unité TPEMS fournit les routines pour sauvegarder et restaurer l'état du gestionnaire de mémoire étendu, afin que vous puissiez

accéder à la mémoire EMS en toute sécurité : *SauveContexteEms* et *RestaureContexteEms*. Pour des explications sur l'écriture de programmes résidents qui utilisent le système Overlays de *Turbo Pascal* 5.0, voir l'appendice D (Utilisation des Overlays avec *Turbo Professionnel*).

Modules et interfaces standard

N'avez-vous jamais tenté accidentellement de charger deux fois le même programme résident ? De nombreux programmes résidents ne peuvent fonctionner selon ce mode et les résultats d'un tel oubli ne sont généralement pas agréables. Néanmoins, TPTSR vous permet de résoudre ce problème potentiel en créant ce que nous appelons une *interface standard*. Ces interfaces permettent à tout programme résident qui les utilise non seulement de savoir s'il a déjà été chargé mais aussi de communiquer avec d'autres programmes résidents (ou *modules*) qui les utilisent. Décrivons rapidement ce dont vous aurez besoin pour résoudre ce problème : affectez à chaque programme résident un nom unique, une chaîne de 5 à 10 caractères. Avant de lancer un programme résident, appelez *ModulePresent* pour voir si ce programme est déjà résident :

```
const
   NomModule : string[6] = 'MONTSR';
...
if ModulePresent(NomModule) then begin
   WriteLn('Déjà installé. Abandon...');
   Halt;
end;
```

Pour que cela fonctionne, il faut installer le module avant de le rendre résident la première fois :

```
InstalleModule(NomModule, Nil);
```

Permet à votre programme de se trouver lui-même et à un autre programme de le trouver.

Pointeurs d'entrée de commande

Pourquoi d'autres programmes auraient-t-ils besoin de vous rechercher ? Vous avez peut-être remarqué le pointeur "Nil" dans l'appel à *InstalleModule* ci-dessus. Le deuxième paramètre de *InstalleModule*, *CmdRoutine* est l'adresse d'une procédure que les autres programmes peuvent appeler. Tout programme peut localiser cette routine au moyen de ce pointeur d'entrée de commande :

```
PointeurEntree := PtrModuleParNom('MONTSR');
if PointeurEntree = Nil then
   {MONTSR n'est pas en mémoire}
else
   PointeurEntree := PointeurEntree^.CmdPtrEnt;
```

Votre *CmdRoutine* devrait être de la forme :

```
procedure MaCmdRoutine(BP : Word); interrupt;
var
   Regs : RegistresInt absolute BP;
begin
   With Regs do begin
      {..gestion d'appels aux fonctions..}
   end;
end;
```

Pourquoi est-ce une procédure d'interruption ? Pour deux raisons. La première est qu'elle peut être appelée facilement en utilisant la routine *EmuleInt* de TPINT. La deuxième est que vous pouvez passer des paramètres en utilisant les registres du CPU comme vous le feriez avec *Intr* et *MsDos*. Par exemple, supposons que *CmdRoutine* exécute un certain traitement si *Regs.AH := 1*. Vous pourriez l'appeler comme ceci :

```
var
   Regs : RegistresInt;
...
Regs.AH := 1;
EmuleInt(Regs, PointeurEntree);
```

Les possibilités d'utilisation sont illimitées. Voir SMACS.PAS pour un exemple.

Déchargement d'un programme résident

Bien souvent les programmes résidents commercialisés ne possèdent pas de commande pour les décharger de la mémoire. Là encore, TPTSR offre cette option à vos utilisateurs. A partir d'une "popup", appelez simplement *DesactiveResid*. Si elle renvoie "True", votre programme résident sera désactivé (déchargé de la mémoire) dès que vous quitterez votre "popup". Si elle renvoie "False", cela signifie qu'il est maintenant très dangereux de décharger le programme résident et il ne le sera pas. Pour une définition de "dangereux", voir les explications de *DesactiveResid*. Comme nous l'avons vu, tous les programmes de démonstration résidents de *Turbo Professionnel* offrent cette option.

Il est également possible d'écrire un programme qui peut se décharger lui-même à partir de la ligne commande du DOS. Nous n'avons pas assez de place pour l'expliquer, mais SMACS.PAS illustre cette technique. Si vous voulez ajouter cette possibilité dans vos propres programmes, étudiez le code source de SMACS, en commençant à la procédure *Install*, à l'endroit où s'effectue la recherche de l'argument de la ligne de commande '-N'.

Autres possibilités d'utilisation des "popups"

Dans les paragraphes précédents, nous avons supposé que vos "popups" étaient destinées à des programmes résidents. Mais ce n'est pas leur seule utilisation. Vous pouvez aussi utiliser des "popups" dans des programmes normaux. Vous pouvez l'utiliser pour afficher une horloge sur l'écran (voir GESTEMPS.PAS comme exemple), pour implémenter une sauvegarde automatique dans un éditeur de texte, les possibilités sont infinies. Souvenez-vous cependant des problèmes de réentrance que vous avons évoqués dans les généralités de l'unité TPINT (chapitre 6) puisque la "popup" sera vraisemblablement appelée pendant l'exécution de votre propre code de programme.

Fonctionnement avec SideKick

SideKick peut entraîner toutes sortes de problèmes pour les autres programmes résidents, mais TPTSR vous aide de deux manières.

Tout d'abord pendant l'initialisation, TPTSR contrôle la présence de *SideKick* en mémoire. Si *SideKick* est présent en mémoire, elle essaie d'adapter la copie résidente de *SideKick* pour permettre de charger en toute sécurité un autre programme résident au-dessus de *SideKick*. Cependant, *SideKick* ne pourra plus s'auto-contrôler contre un éventuel deuxième chargement. Nous admettons que cela n'est pas très satisfaisant, et si vous voulez mieux protéger votre programme résident, vous pouvez simplement désactiver la directive de compilation conditionnelle *'ThwartSideKick'*, qui se trouve dans TPDEFINE.INC, et recompiler TPTSR.

Deuxièmement, si la présence de *SideKick* n'est pas détectée, et que TPTSR ne puisse pas trouver les adresses à modifier, une variable globale appelée *SideKickPresent* sera positionnée et vous pourrez contrôler cette variable avant d'installer le programme résident si vous voulez afficher un message d'erreur. Le contrôle de cet indicateur est facultatif, mais TPTSR refusera l'installation de votre programme résident si *SideKickPresent* vaut "True".

TPTSR peut également détecter la présence de *SideKickPlus*, mais laissera l'indicateur *SideKickPresent* positionné à "False" s'il est trouvé. Vous pourrez toujours charger en toute sécurité un programme résident après *SideKickPlus*.

Fonctionnement en toute sécurité

Avant d'appeler une "popup" devant accéder aux services du DOS, le répartiteur de tâches prend plusieurs précautions pour garantir l'environnement le plus sûr possible : (1) le niveau actuel d'interruption du DOS est sauvegardé, puis suspendu pour garantir que la frappe de (CTRL)-(BREAK) ne détruira pas la "popup" ; (2) de la même façon, les interruptions

$1B et $23 pointent toutes les deux sur les instructions IRET ; (3) l'adresse de transfert disque actuelle et le préfixe de segment de programme sont sauvegardés et ceux qui étaient actifs lorsque le programme résident a été chargé sont restaurés (4). Comme nous l'avons déjà vu, un gestionnaire d'erreurs critiques du DOS est mis en place pour permettre aux "popups" de détecter et de gérer ces erreurs. Lorsque la "popup" rend le contrôle au répartiteur, le système est restauré à l'état.

Notes techniques pour les "hackers"

TPTSR surveille toujours les interruptions suivantes : $05 (Impression d'Ecran), $08 (Timer), $09 (Clavier), $13 (Entrée/Sortie disque), $16 (Lecture clavier), $25 (Lecture d'un secteur sur disque), $26 (Ecriture d'un secteur sur disque) et $28 (Multi-tâches DOS). L'interruption $09 sert à contrôler les hotkeys. Lorsqu'une hotkey est pressée, le compteur d'horloge prend la valeur 36 (2 secondes). Les codes d'interruption $08 et $28 représentent le répartiteur de tâches : si le compteur d'horloge n'est pas à 0, ils vérifieront si une "popup" peut être lancée en toute sécurité et si oui, appeleront la "popup" concernée. Les autres interruptions sont simplement filtrées pour garantir que les "popups" ne seront pas invoquées pendant que l'une de ces interruptions est en activité. Si la "popup" fonctionne sous OS/2, 4 autres interruptions sont également filtrées : $10 (Gestion d'Ecran), $14 (E/S RS232) et $17 (Sortie Imprimante). Voila suffisamment d'informations pour vous donner une certaine idée de la façon dont les "popups" sont gérées. Si vous voulez en savoir plus, vous devrez étudier TPTSR.ASM.

TPTSR : Déclarations

Types

```
PtrIfc = ^EnregIfc;
```

Pointeur sur *EnrIfc*

```
EnregIfc = record
         NomPtr : ^String;
         Version : Word;
         CmdPtrEnt : Pointer;
         IfcPreced,
         IfcSuiv : PtrIfc;
      end;
```

Enregistrement contenant des données sur un module particulier. *NomPtr* pointe sur le nom du module. *Version* est le numéro de version de l'interface. Si *CmdPtrEnt* est différente de nil, cette variable pointe sur la routine qui peut être appelée par d'autres programmes. *IfcPreced* et *IfcSuiv* pointent sur les entrées précédentes et suivantes dans la liste chaînée des modules.

Constantes

```
Int5H = 1;
Int8H = 2;
Int9H = 3;
Int10H = 4;
Int13H = 5;
Int14H = 6;
Int16H = 7;
Int17H = 9;
Int25H = 9;
Int26H = 10;
Int28H = 11;
Int33H = 13;
```

Ce sont des codes d'identification des routines d'interruption utilisées par le gestionnaire d'interruptions de TPTSR (l'interruption 12 est utilisée par TPMACRO).

```
MaxPopups = 8;
```

Nombre maximum de routines "popup".

```
SideKickPresent : Boolean = False;
```

Positionné à "True" dans le bloc d'initialisation de TPTSR si *SideKick* est déjà chargé.

Variables

```
PtrHorlogePop : ^word;
```

Pointe sur une variable cachée dans TPTSR qui indique que vous essayez d'activer un programme résident. La valeur de *PtrHorlogePop^* indique au gestionnaire de popup le nombre de fois où il doit tenter d'activer la routine popup (toutes les 18 fois par secondes). Cette variable est réservée aux utilisations internes.

```
PopupActive : array[1..MaxPopups] of Boolean;
```

Ces indicateurs sont utilisés pour mémoriser les popups actives. Ils sont réservés à des usages internes.

```
PtrPopupsActives : ^Boolean ;
```

Pointe sur une variable cachée qui indique si les popups sont actives. Cette variable est réservée aux usages internes.

```
PtrAppelPopup : ^Byte;
```

Pointe sur une variable cachée qui contient le code d'identification d'une popup en attente si *PtrHorlogePop^* est différent de zéro. Cette variable est réservée aux usages internes.

```
IfcActuel : EnregIfc;
```

Enregistrement contenant des données qui concernent un module particulier.

ChangeHotKey

Déclaration

```
function ChangeHotKey(AncHotKey, NouvHotKey : Word) : Boolean;
```

Objet

Donne à la hotkey *AncHotKey* la valeur de *NouvHotKey*.

Commentaires

Cette routine vous permet de modifier une hotkey pour une "popup" particulière. Il faut cependant que la partie "code de balayage" de la nouvelle hotkey ne soit pas en conflit avec la hotkey d'une autre "popup".

Par exemple, vous pourriez changer une hotkey valant (ALT)-(TAB) en (CTRL)-(TAB), mais vous ne pourrez pas la changer en (CTRL)-( ) si une autre "popup" a déjà (ALT)-( ) comme hotkey.

Exemple

```
const
   AncHotKey = $0829; {Alt-~}
var
   NouvHotKey : Word;
begin
   NouvHotKey := $040F; {Ctrl-Tab}
   if not ChangeHotKey(AncHotKey, NouvHotkey) then
      {la nouvelle hotkey est déjà utilisée}
end.
```

Modifie en (CTRL)-(TAB) la hotkey pour la "popup" associée à (ALT)-~.

DefinitPop

Déclaration

```
function DefinitPop(HotKey : Word; Routine, PtrPile : Pointer;
                    ServicesDos : Boolean) : Boolean;
```

Objet

Définit une routine "popup" associée à une hotkey particulière.

Commentaires

Contrairement à *DefinitProcPop,* cette routine est utilisée pour définir une "popup" qui peut être activée par une hotkey. *Routine* est un pointeur sur la "popup". *PtrPile* pointe sur la fin de la pile utilisée par la "popup". *ServicesDos* indique si la "popup" doit accéder ou non aux services du DOS. *DefinitPop* renvoie "False" s'il n'y a plus de code d'identification de "popup" disponible.

Exemples

```
const
   HotKeyPrincipale : Word = $0829; {Alt-'~'}
   HotKeyLecture    : Word = $0839; {Alt-' '}
var
   PileLecture : array[1..2048] of byte;
...
if not DefinitPop(HotKeyLecture, @PopupLecture, @PileLecture[1], True)
then
   Abandon(PopupImpossible);
```

Incorrect ! '@PileLecture[1]' est l'adresse du début de la pile et non de la fin.

```
if not DefinitPop(HotKeyLecture, @PopupLecture,
                  @PileLecture[SizeOf(PileLecture)], True) then
   Abandon(PopupImpossible);
```

Correct.

```
if not DefinitPop(HotKeyPrincipale, Ptr(CSeg,Ofs(PopupPrincipale)),
                  Ptr(SSeg, SPtr), True) then
   Abandon(PopupImpossible);
```

Peut-être correct, mais ce qui suit est mieux :

```
if not DefinitPop(HotKeyPrincipale, @PopupPrincipale, Ptr(SSeg, SPtr),
True) then
   Abandon(PopupImpossible);
```

Voir aussi

DefinitProcPop (TPTSR)

DesactivePop (TPTSR)

DefinitProcPop

Déclaration

```
function DefinitProcPop(var CodeIdent : Byte;
                        Routine, PtrPile : Pointer) : Boolean;
```

Objet

Définit une routine qui peut être déclenchée par l'appel de *InitHorlogePop*.

Commentaires

Contrairement à *DefinitPop*, cette routine est utilisée pour définir une "popup" qui peut être activée par l'appel de *InitHorlogePop*. *Routine* est un pointeur sur la "popup". *PtrPile* pointe sur la fin de la pile utilisée par la "popup". Il est supposé que la "popup" accède aux services du DOS. *CodeIdent* est un numéro assigné à la "popup" par *DefinitProcPop*. Ce nombre est nécessaire afin d'appeler *InitHorlogePop*. *DefinitProcPop* renvoie "False" s'il n'y a plus de code d'identification de "popup" disponible.

Exemples

```
var
   IdentPop : Byte;
   PileHorloge : array[1..512] of byte;
   ...
if not DefinitProcPop(IdentPop, @AfficheHorloge, @PileHorloge[1]) then
   Abandon(MessErreur);
```

Incorrect ! '@PileHorloge[1]' est l'adresse de début de la pile et non de fin.

```
if not DefinitProcPop(IdentPop, @AfficheHorloge,
      @PileHorloge[SizeOf(PileHorloge)]) then
Abandon(MessErreur);
```

Correct.

Voir aussi

DefinitPop (TPTSR)
InitHorlogePop (TPTSR)
SupprimeProcPop (TPTSR)

DesactivePop

Déclaration

```
procedure DesactivePop(HotKey : Word);
```

Objet

Désactive la routine "popup" associée à une hotkey.

Commentaires

Cette routine vous permet de désactiver une "popup" définie par *DefinitPop*. Son code d'identification peut alors être utilisé pour d'autres "popups".

Exemple

```
const
   MaHotKey = $0439;
...
DesactivePop($0829);
DesactivePop(MaHotKey);
```

Voir aussi

DefinitPop (TPTSR)

SupprimeProcPop (TPTSR)

DesactiveResid

Déclaration

```
function DesactiveResid : Boolean;
```

Objet

Désactive un programme résident en restaurant les vecteurs d'interruption et en libérant la mémoire.

Commentaires

Cette routine vous permet de désactiver (ou décharger) un programme résident dans la plupart des cas. Avant de désactiver le programme, elle vérifie si un autre programme a réservé l'un des vecteurs d'interruption capturés par le programme. Si c'est le cas, la désactivation du programme résident est dangereuse et la fonction retournera "False". Sinon, elle restaurera tous les vecteurs d'interruptions qui ont été capturés et libérera tous les programmes de la mémoire. Elle n'arrête pas réellement le programme. Si *DesactiveResid* renvoie "True", votre programme sortira simplement de la "popup" normalement.

Avertissement : *DesactiveResid* ne doit pas être appelée à partir d'une popup qui n'a pas accès aux services du DOS.

Exemples

```
begin
   {.. début de la popup..}
   if DesactiveResid then
      Halt;
   {.. fin de la popup..}
end.
```

Incorrect! Vous ne devez pas utiliser *Halt* à l'intérieur d'une popup.

```
begin
   {.. début de la popup..}
   if DesactiveResid then
      Exit;
   {.. fin de la popup..}
end;
```

Correct.

Voir aussi

SupprimePops (TPTSR)

EnleveModule

Déclaration

```
procedure EnleveModule;
```

Objet

Enlève le module de la liste chaînée des modules.

Commentaires

Cette routine enlève simplement le module actuel de la liste chaînée des modules. Elle est destinée à l'origine pour les utilisations internes (*DesactiveResid* par exemple).

Voir aussi

DesactiveResid (TPTSR) InstalleModule (TPTSR)

IndicDosActif

Déclaration

```
function IndicDosActif : Byte;
```

Objet

Renvoie la valeur courante de l'indicateur d'occupation du DOS.

Commentaire

Cette routine qui est appelée depuis une popup, indique si le DOS était actif ou non au moment où la popup a été activée. Le DOS n'était pas actif si cette valeur vaut 0, et le DOS était actif dans le cas contraire. En supposant que la popup ait été déclarée pour utiliser les services du DOS, la valeur retournée sera différente de 0 seulement si la popup a été activée pendant que l'utilisateur se trouvait sous le DOS.

Exemple

```
if (IndicDosActif <> 0) then begin
   PlaceChaine(^['TURBO.EXE'^M);
   Exit;
end;
```

En supposant que ces instructions sont exécutées depuis une popup, le buffer clavier contiendra un (ESC), les lettres TURBO.EXE, et un retour chariot, puis le contrôle sera rendu à l'invite du DOS.

IndicErrDos

Déclaration

```
function IndicErrDos : Byte;
```

Objet

Renvoie la valeur courante de l'indicateur d'erreurs critiques du DOS.

Commentaire

Cette routine est appelée depuis une routine popup et indique si une erreur critique du DOS existait ou non au moment où la popup a été activée. Une erreur critique du DOS existait si cette valeur vaut 0 et aucune erreur n'existait dans le cas contraire.

Notez qu'avec certaines versions du DOS, cette valeur pourrait être égale à celle renvoyée par *IndicDosActif*, et n'avoir aucune signification. Explication : examiner la routine *InitTsrPtrs* dans TPTSR.ASM.

InitHorlogePop

Déclaration

```
procedure InitHorlogePop(CodeIdent : Byte; TempsLimite : Word);
```

Objet

Initialise le compteur d'horloge pour la procédure concernée, efface les interruptions et rend le contrôle.

Commentaires

Cette routine demande au répartiteur de tâches d'attendre l'apparition d'un événement significatif pour appeler la "popup" associée au code d'identification. Si le *TempsLimite* du compteur d'horloge est dépassé avant qu'un événement ne se soit produit, la "popup" ne sera pas appelée (il y a 18.2 tops d'horloge par seconde).

Très important : aucun contrôle n'est effectué par *InitHorlogePop*, le code d'identification doit être absolument un numéro valide (*1..MaxPopups*) renvoyé par *DefinitProcPop*.

Exemples

```
InitHorlogePop(1, 36);
```

Incorrect. Le paramètre *CodeIdent* ne peut être une constante même s'il y a une seule "popup" appelée dans votre programme.

```
InitHorlogePop(IdentPopup, 36);
```

Correct. Cela signifie "essayez, pendant les deux secondes qui suivent, d'activer la "popup" associée à *IdentPopup* (retourné par *DefinitProcPop*).

Voir aussi

DefinitProcPop (TPTSR)

InitPops

Déclaration

```
procedure InitPops;
```

Objet

Initialise les structures de données d'un programme résident.

Commentaires

Cette procédure doit être appelée avant toute autre routine "popup". Puisqu'elle est appelée automatiquement quand le programme démarre, vous n'avez pas besoin de l'appeler à moins que vous ne réinstalliez des "popups" après avoir précédemment appelé *SupprimePops*.

Exemple

```
SupprimePops;
{..suite..}
InitPops;
```

Voir aussi

SupprimePops (TPTSR)

InstalleModule

Déclaration

```
procedure InstalleModule(var NomModule : string;
                         CmdRoutine : Pointer);
```

Objet

Installe ce programme comme un module résident auquel peuvent accéder d'autres programmes.

Commentaires

NomModule est une chaîne qui identifie un module résident. *CmdRoutine*, s'il est différent de Nil, doit pointer sur une routine de gestion de pseudo-interruption que les autres programmes peuvent appeler en utilisant *EmuleInt*. *InstalleModule* ajoute le module à la liste chaînée des modules résidents qui utilisent l'interface standard, et de ce fait, permet à n'importe quel module de trouver une copie de lui-même, ainsi que les autres modules, en appelant *ModulePresent*. Le module peut être enlevé de la liste en appelant *EnleveModule*.

Exemple

```
NomTSR : string[6] = 'MONTSR';
...
InstalleModule(NomTSR, Nil);
```

Voir aussi

EmuleInt (TPTSR)
EnleveModule (TPTSR)
ModulePresent (TPINT)

ModulePresent

Déclaration

```
function ModulePresent(var NomModule : string) : Boolean;
```

Objet

Renvoie "True" si *NomModule* est installé.

Commentaires

Cette fonction vous permet de vérifier si un module particulier est installé ou non. Vous pouvez l'utiliser pour éviter de charger votre propre module résident deux fois ainsi que pour vérifier la présence d'un autre programme (tel que SMACS) qui peut être appelé pour exécuter des fonctions spéciales via son pointeur d'entrée de commande.

Exemple

```
const
   NomTSR : string[6] = 'MONTSR';
   AutreTSR : string[10] = 'SUPER MACS';
...
if ModulePresent(NomTSR) then begin
   WriteLn('Déjà en mémoire. Abandon...');
   Halt(1);
end;
if not ModulePresent(AutreTSR) then begin
   WriteLn('SMACS n''est pas en mémoire? Abandon...');
   Halt(1);
end;
```

Voir aussi

PtrModuleParNom (TPTSR)

ParagraphesAConserver

Déclaration

```
function ParagraphesAConserver : Word;
```

Objet

Renvoie le nombre de paragraphes utilisés par le code, les données, la pile et le tas.

Commentaires

Cette routine calcule la quantité totale de mémoire (en paragraphes) utilisée par votre programme. Son usage réel est de vous aider à calculer la valeur à passer à *TermineEtResteResident* (paramètre *ParagraphesAConserver*).

Exemples

```
if not TermineEtResteResident(ParagraphesAConserver, 0) then {erreur}
```

Lance un programme résident sans place supplémentaire sur le tas

```
if not TermineEtResteResident(ParagraphesAConserver + 1024, 0)
then {erreur}
```

Lance un programme résident avec 16K (1,024 paragraphes * 16 octets/paragraphe) de place disponible sur le tas.

Voir aussi

TermineEtResteResident (TPTSR)

PopupsOff

Déclaration

```
procedure PopupsOff;
```

Objet

Désactive les "popups".

Commentaires

Cette routine permet d'éviter les conflits entre les "popups" dans les programmes qui en contiennent plusieurs. Les "popups" peuvent être réactivées avec *PopupsOn*.

Exemple

```
{$F+}
procedure MaPopup(var Regs : Registers);
begin
   PopupsOff;
   {..corps de la popup...}
   PopupsOn;
end;
{$F-}
```

Les autres "popups" sont verrouillées tant que *MaPopup* n'est pas terminée.

PopupsOn

Déclaration

```
procedure PopupsOn;
```

Objet

Active les "popups".

Commentaires

Cette routine active les "popups". Avec les programmes résidents, elle peut être appelée juste avant de lancer le programme résident. Utilisée conjointement à *PopupOff*, elle permet également d'éviter les conflits entre des programmes qui contiennent plusieurs "popup".

Exemples

```
PopupsOn;
if not TermineEtResteResident(ParagraphesAConserver, 0) then
   {Erreur};
```

Séquence standard dans un programme résident : autorise les popups puis reste résident.

```
{$F+}
procedure MaPopup(var Regs : Registers);
begin
   PopupsOff;
   {..corps de la popup...}
   PopupsOn;
end;
{$F-}
```

Les autres "popups" sont verrouillées tant que *MaPopup* n'est pas terminée.

PtrModuleParNom

Déclaration

```
function PtrModuleParNom(var NomModule : string) : PtrIfc;
```

Objet

Renvoie un pointeur sur *EnregIfc* pour le module appelé *NomModule* ou renvoie Nil.

Commentaires

Bien que sensiblement équivalente à *ModulePresent,* l'objet principal de cette routine est de vous permettre de localiser le pointeur d'entrée de commande pour un autre module.

Exemple

```
const NomSmacs : string[10] = 'SUPER MACS';
var P : PtrIfc;
    PointeurEntree : Pointer;
begin
   P := PtrModuleParNom(NomSmacs);
   if P <> Nil then PointeurEntree := P^.CmdPtrEnt
   else PointeurEntree := Nil;
end.
```

Voir aussi

ModulePresent (TPTSR)

ResultInt24

Déclaration

```
function ResultInt24 : Word;
```

Objet

Renvoie le dernier résultat du gestionnaire de INT $24.

Commentaires

Cette routine est fonctionnellement identique à *ResultInt24* de l'unité TPINT24. La seule différence est que le gestionnaire d'erreurs critiques de TPTSR est installé seulement lorsqu'une "popup" est activée. Le gestionnaire d'erreurs ne peut pas être désactivé. Voir les explications de *ResultInt24* (TPINT24) pour des exemples.

Voir aussi

ResultInt24 (TPINT24)

SupprimePops

Déclaration

```
procedure SupprimePops;
```

Objet

Enlève toutes les "popups" du système et libère tous les vecteurs d'interruption associés.

Commentaires

Cette routine est destinée à l'origine pour les utilisations internes. Elle est appelée automatiquement lorsqu'un programme se termine. Remarquez qu'elle n'a pas la même fonction que *DesactiveResid* qui termine effectivement un programme résident.

Voir aussi

DesactiveResid (TPTSR) InitPops (TPTSR)

SupprimeProcPop

Déclaration

```
procedure SupprimeProcPop(CodeIdent : Byte);
```

Objet

Supprime une routine "popup" définie par *DefinitProcPop*.

Commentaires

Cette routine vous permet de supprimer une "popup" définie par *DefinitProcPop*. Son code d'identification peut alors être utilisé pour les autres "popups".

Exemple

```
var
   IdentPop : Byte;
...
if not DefinitProcPop(IdentPop, @MaPopup, @MaPile[SizeOf(MaPile)])
then {Erreur};
{..Suite..}
SupprimeProcPop(IdentPop);
```

Voir aussi

DefinitProcPop (TPTSR) DesactivePop (TPTSR)

TermineEtResteResident

Déclaration

```
function TermineEtResteResident(ParasAConserver : Word;
                                CodeSortie : Byte) :  Boolean;
```

Objet

Termine et reste résident en libérant toute la mémoire sauf *ParasAConserver*.

Commentaires

Cette routine (1) libère toute la mémoire allouée au programme sauf *ParasAConserver* (nombre de paragraphes à conserver, habituellement calculé par *ParagraphesAConserver*) ; (2) définit la limite du tas pour que la mémoire puisse être allouée dynamiquement une fois le programme résident lancé ; (3) restaure le vecteur d'interruption capturé à l'exécution de Turbo et (4) appelle la fonction *Keep* de l'unité DOS ($31), en lui passant *ParasAConserver* et *CodeSortie* (presque toujours égal à 0) en paramètres. *TermineEtResteResident* renvoie "False" s'il n'est pas possible de libérer la mémoire pour le programme.

Exemples

```
if not TermineEtResteResident(ParagraphesAConserver, 0) then {Erreur}
```

Lance un programme résident sans place supplémentaire sur le tas.

```
if not TermineEtResteResident(ParagraphesAConserver + 1024, 0)
then {Erreur}
```

Lance un programme résident avec 16K (1,024 paragraphes * 16 octets/paragraphe) de place disponible sur le tas.

Voir aussi

ParagraphesAConserver (TPTSR) PopupsOn (TPTSR)

TP8087

Comme nous l'avons vu, l'unité TP8087 est destinée principalement aux programmes résidents qui utilisent un co-processeur mathématique (8087) pour des calculs sur des nombres réels (Single, Double, Extended, Comp). Elle contient une fonction (*Presence8087*) qui permet de tester la présence du composant mais elle est rarement utilisée puisque *Turbo Pascal* 4.0 génère un code d'initialisation qui teste sa présence lorsque vous compilez avec la directive {$N+} et *Turbo Pascal* 5.0 émule la virgule flottante. *Sauve8087* et *Restaure8087*, cependant, sont très importantes si votre programme résident doit utiliser le co-processeur. Bien que le répartiteur de tâches de TPTSR protège les registres du CPU, il ne sauvegarde pas les registres du co-processeur ; aussi, si votre "popup" utilise des réels 8087, la routine principale doit avoir l'aspect suivant :

```
{$F+}
procedure PointEntreePopup;
var
   SauveRegs8087 : SauveBuffer8087;
begin
   Sauve8087(SauveRegs8087);
   {... suite des routines popups...}
   Restaure8087(SauveRegs8087);
end;
{$F-}
```

Ces deux procédures vérifient que les "popups" utilisant le co-processeur ne causeront pas de problème si le programme est interrompu. Par défaut, *Turbo Pascal* produira une erreur fatale d'exécution toutes les fois qu'un calcul numérique avec le co-processeur est erroné. Division par zéro, dépassement numérique, racine carrée d'un nombre négatif, ces erreurs et bien d'autres arrêteront votre programme. Même l'unité TPEREXEC ne peut être d'aucune aide puisqu'au moment où elle prendra le contrôle de cette erreur, la pile interne 8087 aura été irrémédiablement endommagée.

Fort heureusement, TP8087 offre les moyens de résoudre ce problème avec deux simples routines. La procédure *Exceptions8087* accepte un paramètre booléen qui générera une procédure d'exception du co-processeur (qui conduira à des erreurs d'exécutions) ou rapportera des erreurs via le mot d'état du co-processeur et les résultats numériques. La deuxième routine, *Erreur8087*, permet de tester le mot d'état du co-processeur pour vérifier si les calculs précédents ont produit une erreur numérique. La routine *Erreur8087* renvoie un résultat identique à celui de la fonction *IoResult* : toutes les valeurs différentes de zéro impliquent une erreur (qui peut être décodée et interprétée) et le processus de lecture de *Erreur8087* réinitialise le registre d'état pour les calculs ultérieurs.

Il est important de noter que TP8087 aura un comportement légèrement différent selon la façon dont elle est compilée. Si elle est compilée avec *Turbo Pascal* 4.0, ou si elle est compilée avec la version 5.0 et la directive de compilation $N-, toutes les routines de l'unité utiliseront les instructions 8087. Si elle est compilée avec la version 5, et avec la directive de compilation $N+, toutes les routines utiliseront l'émulateur virgule flottante de l'unité SYSTEM. Aussi, la routine *Sauve8087* par exemple, sauvegardera les registres du coprocesseur ou les registres de l'émulateur, selon l'installation du coprocesseur mathématique. A la suite de ce chapitre, toutes les références à 8087 sont équivalentes à des références à l'émulateur virgule flottante de *Turbo Pascal.*

Types

```
SauveBuffer8087 =
      record
            Controle : Word;        {mot de contrôle}
            Statut : Word;          {mot d'état}
            Sommet : Word;          { sommet de pile}
            PtrInst : LongInt;      {pointeur d'instructions}
            PtrOp : LongInt; {pointeur de données}
            Regs8087 : array[0..7] of Extended;   {pile}
      end;
```

Utilisé pour enregistrer des données nécessaires pour sauvegarder et ultérieurement restaurer l'état du co-processeur mathématique en utilisant *Sauve8087* et *Restaure8087.*

Erreur8087

Déclaration

```
function Erreur8087 : Word;
```

Objet

Renvoie (et efface) le statut d'erreur du 80x87.

Commentaires

Cette fonction peut être utilisée pour contrôler les erreurs après des opérations arithmétiques exécutées avec le co-processeur mathématique. Elle est utile si vous avez programmé le co-processeur de telle façon qu'il n'utilise pas les interruptions d'exceptions avec la procédure *Exceptions8087*. Elle renvoie les six bits de poids faible du mot d'état du co-processeur qui indiquent les conditions d'erreurs et réinitialise ces indicateurs pour autoriser les calculs ultérieurs.

Si la fonction renvoie 0, aucune erreur ne s'est produite. Sinon, vous devez déterminer quelle exception a été générée en testant chaque bit dans le mot retourné :

Bit	Valeurs	Significations
5	32	Perte de précision (résultat inexact - arrondi)
4	16	Dépassement vers les valeurs basses
3	8	Dépassement de capacité
2	4	Division par 0
1	2	Opérande dénormalisé (résultat trop petit, la mantisse contient des 0 en tête)
0	1	Opération non valide (telle que SQRT ou LN sur des nombres négatifs)

Remarque : Si ces deux erreurs ne vous concernent pas, vous pouvez simplement masquer ces bits en effectuant un AND logique entre le résultat de la fonction et $0D.

Exemple

```
Exceptions8087(False);
...
Etendu3 := Etendu1 * Etendu2;
CtrlErreur := Erreur8087 {and $0D};
```

L'appel à *Erreur8087* charge *CtrlErreur* avec le contenu du mot d'état du co-processeur et efface ces indicateurs d'erreurs.

Voir aussi

Exceptions8087 (TPTSR)

Exceptions8087

Déclaration

```
procedure Exceptions8087(On : Boolean);
```

Objet

Active ou désactive les conditions d'interruption.

Commentaires

Cette procédure permet de programmer le co-processeur mathématique pour produire des conditions d'interruptions qui généreront des erreurs à l'exécution ou pour traiter des erreurs arithmétiques en utilisant le mot d'état du co-processeur. Puisque le 8087 produit des conditions d'interruptions par défaut, la routine fournira le moyen de reprendre le contrôle après d'autres erreurs d'exécutions fatales. Lorsque vous avez désactivé les conditions d'interruption, vous devez vérifier les erreurs vous-même en testant *Erreur8087*.

Remarque : Les instructions FINIT et FNINIT utilisées par les routines telles que *Presence8087* lors du test de la présence du 80x87, reprogrammeront le co-processeur pour générer des conditions d'interruptions. Si vous exécutez une commande du DOS (en utilisant *ExecDos* ou des routines *Exec* de Turbo), après avoir désactivé les conditions d'interruption, vous devrez appeler *Exceptions8087* après l'appel de *Exec* au cas où le programme que vous exécutez ait reprogrammé le co-processeur.

Exemples

```
Exceptions8087(False);
...
Etendu3 := Etendu1 * Etendu2;
CtrlErreur := Erreur8087;
```

L'appel à *Exceptions8087* désactive les conditions d'interruptions pour pouvoir appeler *Erreur8087* après les opérations arithmétiques.

```
Exceptions8087(False);
...
Code := ExecDos('', True, Nil);
Exceptions8087(False);
```

Le deuxième appel à *Exceptions8087* garantit que le co-processeur est programmé correctement après l'appel de *Exec*.

Voir aussi

Erreur8087 (TP8087)

ExecDos (TPDOS)

Presence8087

Déclaration

```
function  Presence8087 : Boolean;
```

Objet

Renvoie "True" si le co-processeur 80x87 est installé.

Commentaires

Cette fonction permet de déterminer si, sur une machine particulière, un co-processeur mathématique est installé. Puisque la bibliothèque d'exécution de Turbo effectue son propre test et refuse de charger un programme qui utilise le co-processeur alors que celui-ci n'est pas installé, la fonction *Presence8087* est rarement nécessaire. Elle est utile dans les programmes d'installation pour configurer des programmes qui nécessitent un co-processeur. Comme nous l'avons vu, cette fonction a un effet secondaire qui réinitialise le co-processeur à son état par défaut et, par ce moyen, rétablit les conditions d'interruptions. Voir la procédure *Exeptions8087*.

Exemple

```
if not Presence8087 then begin
   WriteLn('Ce programme nécessite un coprocesseur arithmétique !');
   Halt(1);
end;
```

Ce code pourrait figurer dans un programme d'installation utilisé pour installer un programme nécessitant un co-processeur mathématique.

Voir aussi

Exceptions8087 (TP8087)

Restaure8087

Déclaration

```
procedure Restaure8087(var SauveBuf : SauveBuffer8087);
```

Objet

Restaure les registres du 80x87 à partir du buffer de sauvegarde.

Commentaires

Cette procédure est destinée à l'origine aux routines "popups" écrites avec TPTSR. Elle permet de restaurer l'état du co-processeur (ses registres et la pile), pour que les opérations arithmétiques exécutées par la "popup" n'affectent pas le programme qui a été interrompu. Chaque appel à *Restaure8087* doit être précédé par un appel à *Sauve8087* qui sauvegarde l'état du co-processeur.

Exemple

```
procedure MaPopup;
var
   SauveBuf : SauveBuffer8087;
begin
   Sauve8087(SauveBuf);
   {... suite des routines popups...}
   Restaure8087(SauveBuf);
end;
```

Restaure8087 est appelée en sortie d'une routine "popup" pour restaurer l'état du co-processeur.

Voir aussi

Sauve8087 (TP8087)

Sauve8087

Déclaration

```
procedure Sauve8087(var SauveBuf : SauveBuffer8087);
```

Objet

Sauve les registres du 80x87 dans un buffer de sauvegarde.

Commentaires

Cette procédure est destinée à l'origine aux routines "popups" écrites avec TPTSR. Elle permet de sauvegarder l'état du co-processeur pour que la "popup" puisse l'utiliser sans affecter le programme qui a été interrompu. Chaque appel à *Sauve8087* doit être suivi par un appel à *Restaure8087* qui restaure l'état du co-processeur.

Exemple

```
{$F+}
procedure MaPopup;
var
   SauveBuf : SauveBuffer8087;
begin
   Sauve8087(SauveBuf);
   {... suite des routines popups...}
   Restaure8087(SauveBuf);
end;
{$F-}
```

Sauve8087 est appelée en entrée de la routine "popup" pour sauvegarder l'état du co-processeur.

Voir aussi

Restaure8087 (TP8087)

TPINT24 : Gestionnaire d'erreurs critiques

TPINT24 offre à vos programmes un gestionnaire d'erreurs critiques du DOS qui s'installe lui-même automatiquement lorsque votre programme démarre et se décharge lui-même lorsqu'il se termine. Bien que la bibliothèque d'exécution de Turbo Pascal 4.0 contienne également un gestionnaire d'erreurs critiques, la méthode qu'il utilise pour renvoyer les codes erreurs peut quelquefois rendre difficile l'interprétation exacte de l'erreur qui s'est produite et Turbo ne vous donne aucun moyen facile de désactiver temporairement le gestionnaire d'erreurs. Pour ces raisons, vous pourrez préférer le gestionnaire d'erreurs de TPINT24.

Pour utiliser TPINT24 dans vos programmes, vous devrez suivre trois étapes. Premièrement, ajoutez TPINT24 dans la clause USES au début de votre programme. Deuxièmement, désactivez le contrôle des erreurs d'entrée/sortie en utilisant la directive de compilation {$I-}. Troisièmement, après chaque opération d'entrée/sortie, qui pourrait entraîner une erreur, ajoutez un appel à la fonction *ResultInt24*. En fait, vous devrez remplacer les appels normaux à *IoResult* par des appels à *ResultInt24*. Elle renvoie un mot dont l'octet de poids fort contient le code de l'erreur critique du DOS et l'octet de poids faible contient le résultat de *IoResult*. Les deux autres routines de l'unité, *Int24On* et *Int24Off*, permettent de décharger temporairement le gestionnaire d'erreurs.

Note technique : Sous DOS 2.x, le gestionnaire d'erreur de TPINT24 demandera au DOS d'ignorer l'erreur critique car c'est le seul moyen qui permette à votre programme de traiter correctement l'erreur. Avec la version DOS 3.x, le gestionnaire d'erreur demandera au DOS d'interrompre l'appel ayant générer l'erreur pour qu'elle soit traitée.

Int24Off

Déclaration

```
procedure Int24Off(RestaureErrDos : Boolean);
```

Objet

Restaure le gestionnaire d'erreurs critiques du DOS de Turbo.

Commentaires

Cette procédure décharge le gestionnaire d'erreurs critiques TPINT24. Si *RestaureErrDos* a la valeur "True", le gestionnaire d'erreurs critiques du DOS est restauré à sa place. Si elle a la valeur "False", c'est le gestionnaire d'erreurs critiques de Turbo qui est restauré.

Voir aussi

Int24On (TPINT24)

Int24On

Déclaration

```
procedure Int24On;
```

Objet

Installe le gestionnaire d'erreurs critiques du DOS.

Commentaires

Cette procédure réinstalle le gestionnaire d'erreurs critiques TPINT24. Il n'est pas nécessaire de l'appeler au début de votre programme puisque c'est automatique.

Voir aussi

Int24Off (TPINT24)

ResultInt24

Déclaration

```
function ResultInt24 : Word;
```

Objet

Renvoie le code d'erreur critique du DOS dans l'octet de poids fort et la fonction *IoResult* dans l'octet de poids faible.

Commentaires

Cette fonction renvoie *IoResult* dans l'octet de poids faible et le code erreur critique du DOS (voir liste ci-dessous) dans l'octet de poids fort. Afin que cette routine fonctionne, le contrôle des entrées/sorties doit être désactivé par la directive de compilation {$I-} et *ResultInt24* doit être contrôlée scrupuleusement après chaque opération d'entrée/sortie qui pourrait générer une erreur.

Remarque : Bien qu'il soit parfois possible de traiter des erreurs d'imprimante (hors ligne, plus de papier, etc) sur certaines machines, la détection de ces erreurs n'est pas garantie, et vous ne devez pas compter dessus.

Codes d'erreurs critiques

$01	Numéro de lecteur non valide
$02	Lecteur non prêt
$03	Commande inconnue
$04	Erreur de CRC (Contrôle de Redondance Cyclique)
$05	Mauvaise longueur d'une demande
$06	Erreur de recherche
$07	Format disque non reconnu
$08	Secteur non trouvé
$09	Plus de papier
$0A	Erreur en écriture
$0B	Erreur en lecture
$0C	Panne générale
$0D	Tentative d'écriture sur une disquette protégée ($00 à $0D)

Exemple

```
Assign(F, NomFichier);
Reset(F);
if ResultInt24 <> 0 then begin
   WriteLn('Erreur à l''ouverture de ', NomFichier);
   Exit;
end;
```

L'appel à *ResultInt24* vérifie si une erreur d'entrée/sortie ou une erreur critique du DOS ne s'est pas produite pendant l'ouverture de *F*.

7 Tri

Généralités

Le tri est une opération couramment exécutée par de nombreux programmes. Il peut être utilisé pour trier des listes d'adresses dans l'ordre alphabétique ou pour effectuer des statistiques sur des ensembles de nombres.

Turbo Professionnel fournit une routine de tri qui est rapide et facile à utiliser. Vous pouvez trier jusqu'à 65535 éléments de données de n'importe quel type. Tous les éléments doivent tenir en mémoire centrale puisque la pagination n'est pas supportée. L'unité TPTRI est très compacte et utilise moins de 2000 octets de code. Elle met en oeuvre l'agorithme de tri rapide (Quicksort) non-récursif qui minimise aussi l'utilisation de la pile.

Si vous connaissez le tri de la *ToolBox Database* de *Borland,* l'utilisation de l'unité TPTRI sera un jeu d'enfant puisque ces deux systèmes sont organisés de la même façon. TPTRI diffère principalement dans son implémentation. Comme nous l'avons vu, TPTRI ne supporte pas la pagination et fournit à la place un tri plus compact et plus rapide, adapté à la plupart des situations. TPTRI traite les chaînes de caractères selon un mode spécial et alloue simplement la longueur réelle plutôt que la longueur maximum. L'utilisation des pointeurs permet à TPTRI de faire un tri complètement dynamique et permet de trier la plupart des types de données dans le même programme.

TPTRI comporte trois routines : *Tri, PlaceElement, PrendElement.* Votre programme appelera *Tri* en premier. *Tri* comporte plusieurs paramètres qui seront décrits en détails. Deux des paramètres décrivent les éléments à trier, leur taille et le nombre maximum attendu. Trois paramètres fournissent des adresses de routines que *Tri* appelera pendant son exécution : une routine qui chargera les éléments dans les structures de données de tri, une autre que *Tri* appelera pour comparer un couple d'éléments et une dernière routine pour renvoyer les résultats triés à votre programme.

Les routines que vous fournirez pour charger et retrouver les éléments appeleront les deux autres fonctions exportées par TPTRI. Pour chaque

élément à trier, votre routine appelera *PlaceElement*. Pour entrer les données pour chaque élément trié, votre routine appelera *PrendElement*.

L'exemple qui suit, met en oeuvre le processus de tri d'un ensemble d'entiers :

```
program TriEntiers;
{Ce programme trie un ensemble aléatoire d'entiers}
uses TpTri;
const NbATrier = 1000;
var Statut : StatutTri;

{$F+}
procedure LitEntiers;
var
   N : Word;
   Valeur : integer;
begin
   for N := 1 to NbATrier do begin
      Valeur := Random(MaxInt);
      if not PlaceElement(Valeur) then
         {Mémoire insuffisante est l'erreur la plus vraisemblable}
         Exit;
   end;
end;

function Inferieur(var X, Y) : boolean;
{..Compare deux entiers}
var
   XI : integer absolute X;
   YI : integer absolute Y;
begin
   Inferieur := (XI < YI);
end;

procedure EcritEntiers;
var
   Valeur : integer;
begin
   While PrendElement(Valeur) do
      WriteLn(Valeur);
end;
{$F-}

begin
   Statut := Tri(NbATrier, SizeOf(Integer), @LitEntiers, @Inferieur,
                 @EcritEntiers);
   case Statut of
      SuccesTri      : WriteLn(ElementsTries, ' éléments triés');
      TriHorsMemoire : WriteLn('Mémoire insuffisante');
      TropElementsATrier : WriteLn('Trop d''éléments à trier');
   end;
end.
```

Consultez les descriptions qui suivent pour plus de détails.

TPTRI et la gestion de la mémoire

Normalement, TPTRI utilise une technique spéciale pour libérer rapidement le tas sans détruire la table des fragments du gestionnaire de mémoire Turbo. Cette routine fonctionne bien, tant que le programme appelant n'a pas besoin d'allouer ou de libérer le tas pendant que le tri est actif, c'est-à-dire dans les procédures utilisateur. Dans ce cas, modifiez le code source de TPTRI. Faites en sorte que cette modification soit simple. Désactivez la directive de compilation conditionnelle *FastDispose* dans TPDEFINE.INC, et recompilez TPTRI.

Erreurs les plus fréquentes

90% des appels que nous avons reçus étaient relatifs aux deux erreurs suivantes :

- les routines utilisateurs appelées par Tri n'avaient pas été déclarées FAR avec {$F+},
- elles étaient locales par rapport à une autre procédure.

Pour que le tri fonctionne correctement, les routines utilisateurs doivent être des procédures globales compilées avec le modèle FAR. Veuillez le vérifier si vous avez des problèmes avec TPTRI.

TPTRI : Routine de tri en mémoire

Cette partie décrit tous les types, variables, constantes, procédures et fonctions qui se trouvent dans la partie interface de l'unité TPTRI.

Types

```
StatutTri =
   (SuccesTri,         {Tri bien terminé}
   TriHorsMemoire,     {Mémoire insuffisante}
   TropElementATrier {Il y a plus d'éléments à trier que demandés}
   );
```

La fonction *Tri* renvoie un résultat de type *StatutTri*. L'application appelante doit contrôler ce résultat.

Variables

```
ElementsTries : Word;
```

Quand le tri est terminé, la variable *ElementsTries* contient le nombre d'éléments réellement triés.

PlaceElement

Déclaration

```
function PlaceElement(var X) : Boolean;
```

Objet

Place un élément à trier dans le tableau de tri.

Commentaires

Appelez *PlaceElement* pour ajouter chaque nouvel élément à la structure interne de TPTRI. Votre application appellera *PlaceElement* à l'intérieur de la routine de chargement dont l'adresse est passée à la fonction *Tri*.

Puisque TPTRI ne connaît pas le type de vos données, le paramètre de *PlaceElement* est un paramètre VAR sans type. Vous ne pouvez donc pas passer une constante ou une expression à *PlaceElement*. Quoi qu'il en soit, le paramètre peut être n'importe quel type de variable, tel qu'un élément numérique simple, un pointeur sur une autre structure ou une chaîne de caractères de longueur variable. TPTRI connaît le nombre d'octets à transférer en fonction de l'appel initial à *Tri*. *PlaceElement* renvoie "True" lorsqu'un élément est additionné correctement au tableau de tri. Si elle renvoie "False", votre application devra arrêter l'ajout des éléments et sortir de la routine de chargement. *PlaceElement* renverra "False" s'il n'y a pas assez de place sur le tas pour un nouvel élément ou si le nombre d'éléments dépasse le maximum défini lors de l'appel à *Tri*.

Exemples

```
while not Eof(F) do begin
   ReadLn(F, S);
   if not PlaceElement(S) then Exit;
end;
```

Cet exemple lit des lignes sur un fichier texte et ajoute chacune d'elle dans le tableau de tri. Si *PlaceElement* renvoie "False", la routine de chargement est terminée.

```
for I := 1 to NbPointeurs do
   if not PlaceElement(P[I]) then Exit;
```

Dans cet exemple, *P* est un tableau de pointeurs sur une autre structure de données, peut-être un enregistrement. Les pointeurs eux-mêmes sont ajoutés au tri au lieu des enregistrements complets (pour économiser la place mémoire). La fonction de comparaison d'éléments déréférencera les pointeurs pour effectuer la comparaison.

Voir aussi

Tri (TPTRI)

PrendElement

Déclaration

```
function PrendElement(var X) : Boolean;
```

Objet

Prend l'élément suivant du tableau trié.

Commentaires

Appelez *PrendElement* pour retrouver chaque élément trié. Votre application appellera *PrendElement* à l'intérieur de la routine de chargement dont l'adresse a été passée à la fonction *Tri*.

Puisque TPTRI ne peut pas connaître le type de vos données, le paramètre VAR de *PrendElement* est sans type. Il est de votre responsabilité de fournir une variable du type approprié. TPTRI connaît le nombre d'octets à retourner en fonction du paramètre passé à la fonction *Tri*.

PrendElement renvoie "True" tant qu'il y a des éléments à retrouver. Une fois qu'elle renvoie "False", le paramètre *X* est indéfini.

Exemple

```
while PrendElement(S) do
   WriteLn(S);
```

Cet exemple écrit simplement les éléments triés (de n'importe quel type simple) à l'écran.

Voir aussi

Tri (TPTRI)

Tri

Déclaration

```
function Tri(Elements : Word, TailleElement : Word;
             PrendElem : Pointer; PlusPetit : Pointer;
             PlaceElem : Pointer) : StatutTri;
```

Objet

Trie un ensemble d'éléments.

Commentaires

Tri est la routine principale de TPTRI. Une application appelera *Tri* avant d'appeler n'importe quelle autre routine. *Tri* est conçue pour être complètement paramétrable à l'exécution. A l'intérieur des limites de la mémoire, elle peut trier jusqu'à 65535 éléments de n'importe quel type en appelant les fonctions définies par l'utilisateur pour comparer, entrer et sortir les éléments. *Tri* conserve tous les éléments en mémoire. Ce choix est plus rapide qu'un tri sur disque et offre une capacité suffisante dans la plupart des cas.

Les deux premiers paramètres passés à *Tri* indiquent combien d'éléments sont attendus et la taille de chaque élément. Le paramètre *Elements* spécifie le nombre maximum attendu. Il n'y a pas de problème s'il y a moins d'éléments réellement utilisés. S'il y a plus d'éléments en réalité, une erreur de tri se produira.

Le paramètre *TailleElement* spécifie le nombre d'octets dans chaque élément à trier. La taille est obtenue en utilisant la fonction SizeOf de Turbo. Par exemple, *SizeOf(pointer)* renvoie 4 et *SizeOf(TypeDonnee)* renvoie le nombre d'octets du type de données. *Tri* fournit un traitement spécial pour les chaînes. Si vous spécifiez une taille d'élément égale à zéro, *Tri* interprétera chaque élément comme une chaîne de Turbo et allouera la mémoire uniquement pour la taille demandée par chaque chaîne.

Les programmes utilisant TPTRI doivent être structurés d'une certaine façon, comme le montre l'exemple présenté au début de ce chapitre. L'application doit définir trois routines dont les adresses sont passées comme des pointeurs à la routine *Tri*.

```
{$F+}
procedure LitElementsATrier;
{soumet les éléments à trier à la routine de tri}
begin
   ...
   if not PlaceElement(x) then
      Exit;
   ...
end;
```

```
procedure UtiliseElementsTries;
{renvoie la liste des éléments triés}
begin
   ...
   While PrendElement(x) do
      {utiliser l'élément d'une manière ou d'une autre}
end;

function Inferieur(var X, Y) : boolean;
begin
   {retourne la valeur de l'expression booléenne : (X < Y)}
end;
{$F-}
```

Remarquez que les trois routines doivent être du même type (function, procedure) et traiter les mêmes paramètres comme sur l'exemple (les noms des routines sont sans importance) et elles doivent être compilées avec l'option FAR. L'adresse de chaque routine est passée à *Tri* dans les trois paramètres pointeurs. Dans cet exemple, *Tri* sera appelée de la façon suivante :

```
Statut := Tri(Elements, TailleElement, @LitElementsATrier, @Inferieur,
              @UtiliseElementsTries);
```

L'opérateur '@' indique au compilateur de passer les adresses des routines à la fonction *Tri*.

Tri appelera *LitElementsATrier* après avoir effectué certains contrôles d'erreurs et certaines tâches d'initialisation. *LitElementsATrier* appelle *PlaceElement* pour placer chaque élément dans la structure de données internes de TPTRI. Lorsque *LitElementsATrier* prend le contrôle, Tri exécute réellement les opérations de tri. Toutes les fois que *Tri* a besoin de comparer deux éléments pour les placer dans l'ordre de tri, elle appelle votre fonction *Inferieur*. La fonction *Inferieur* compare les éléments de la façon souhaitée en fonction du type de données et renvoie "True" si le premier élément *X* est plus petit que le second élément *Y*. Une fois que le tri est terminé, *Tri* appelle votre routine *UtiliseElementsTries* qui appelle *PrendElement* pour retourner chaque élément dans l'ordre. Lorsque *UtiliseElementsTries* prend le contrôle, *Tri* libère la mémoire qui a été utilisée et renvoie un statut de tri au programme appelant. Voir le type *StatutTri* pour les codes de tri utilisés.

Exemples

Voir le programme exemple au début de ce chapitre.

8 Allocation mémoire

Généralités

Beaucoup de choses ont déjà été écrites sur la limite des 640 K imposés par MS-DOS et beaucoup d'entre nous ont été confrontés à ce problème. Des logiciels sophistiqués ont besoin de toute la mémoire pour le code et surtout pour les données mais que pouvez-vous faire lorsque la mémoire disponible est insuffisante ? Heureusement, il existe deux alternatives à la RAM normale.

L'une de ces possibilités, la *mémoire étendue*, est disponible seulement sur les machines '286' et '386'. La mémoire étendue existe au-dessus de la limite de 1 mégaoctet et est accessible seulement en plaçant ces machines en mode protégé.

L'autre possibilité est la mémoire EMS (Expanded Memory Specification) disponible sur toute machine compatible IBM équipée d'une carte supportant la spécification EMS. La mémoire EMS est gérée par un logiciel (Memory Manager) et plusieurs sociétés ont développé des drivers qui permettent à la mémoire étendue d'émuler la mémoire EMS. Malheureusement, *Turbo Pascal* comme la plupart des compilateurs ne vous permet pas d'utiliser la mémoire étendue ou la mémoire EMS pour enregistrer les données. *Turbo Professionnel* vous le permet.

L'unité TPEMS contient de nombreuses routines pour travailler avec la mémoire EMS. Elle permet de déterminer si la mémoire EMS est installée ou non et, si oui, la taille de la mémoire disponible actuellement. Vous pouvez allouer et libérer de la mémoire, et vous pouvez également sauvegarder et restaurer l'état ("contexte") du gestionnaire de mémoire EMS lorsque vous devez accéder à la mémoire EMS, à partir d'un programme résident par exemple.

Vous aurez probablement noté que TPEMS est adapté aux versions 3.x ou supérieures, des spécifications de mémoires étendues. L'unité ne supporte actuellement aucune des nouvelles fonctionnalités de EMS 4.0.

La mémoire EMS est allouée par blocs de 16 K appelés *pages*. Elle est très pratique pour enregistrer des tableaux, des listes chaînées, des arbres

binaires de taille moyenne, mais relativement moins adaptée pour enregistrer des petites quantités de données qui doivent être atteintes rapidement par un mécanisme normal. Si vous n'avez pas de gros besoins en mémoire dynamique, il vaut mieux allouer la mémoire en RAM normale en utilisant *New* ou *GetMem*. Bien que la mémoire EMS ne soit pas très adaptée à l'enregistrement de grands tableaux, *Turbo Professionnel* fournit une solution dans l'unité TPTBMEXP. En utilisant les routines de cette unité, vous pouvez facilement travailler avec des tableaux allant jusqu'à 8 mégaoctets. Voir les explications de TPTBMEXP au chapitre 9, section C, pour plus de détails.

De la même façon, l'unité TPMEMEXT fournit tous les outils pour utiliser la mémoire étendue afin d'enregistrer vos données. Ces routines permettent d'accéder à la mémoire étendue sous DOS 3.x ou 4.x sans interférence avec VDISK, avec RAMDRIVE.SYS de MicroSoft ou avec tout autre programme compatible avec VDISK et RAMDRIVE. Pour accéder à la mémoire, vous devrez suivre ces étapes :

1. Trouver la taille de la mémoire étendue disponible en appelant *MemExtDisp*.
2. Si le total de mémoire disponible est suffisant, utiliser *AlloueMemExt* pour allouer la place mémoire dont vous avez besoin.
3. Une fois que la mémoire a été allouée, utiliser *DeplaceMemExt* pour déplacer vos données à partir de la mémoire RAM normale vers la mémoire étendue et vice et versa.

Comme vous l'avez déjà deviné, TPMEMEXT contient la contre-partie des routines internes de *Turbo Pascal* pour gérer la mémoire étendue : *MemAvail, GetMem, FreeMem, Mark, Release,* et *Move*. Bien que les noms soient similaires, il existe d'importantes différences entre elles et vous devez garder à l'esprit ces quelques recommandations lorsque vous utilisez ces routines.

La première et la plus importante est que la mémoire étendue doit être allouée et libérée en ordre séquentiel. Si vous allouez trois blocs de mémoire et libérez le bloc du milieu en appelant *DeallouemMemExt*, toute la mémoire au-dessus du premier bloc sera indisponible (les raisons de cette limitation seront évidentes si vous étudiez le code source de TPMEMEXT).

La seconde importante recommandation à ne pas oublier est que les adresses de la mémoire étendue sont utilisées avec une notation *absolue* et non une notation *segmentée*. Par exemple, l'adresse segmentée $0040:$0012 correspond à l'adresse absolue $00000412. Le léger désavantage de cette méthode est que lors du déplacement de données entre la mémoire normale et la mémoire étendue vous devez convertir les adresses segmentées en adresses absolues ou *pointeurs étendus* en appelant la fonction *PtrEnEtendu*.

L'énorme avantage de cela est que les pointeurs étendus ne sont rien d'autres que des entiers longs qui vous permettent de créer des tableaux plus grands que 64 K dans la mémoire étendue en utilisant les entiers longs comme index de tableau. Par exemple, si PE est un pointeur étendu sur l'adresse 0 d'un tableau d'octets, alors vous pouvez calculer l'adresse du nième élément (PEn) de la façon suivante :

```
PEn := PtrEtendu(LongInt(PE) + (N*SizeOf(Element)));
```

Troisièmement, *AlloueMemExt* permet d'allouer la mémoire par blocs de 1 K (1024 octets). Comme pour la mémoire EMS, la mémoire étendue est moins intéressante pour les enregistrements de type général, mais cela signifie aussi que vous pouvez allouer plusieurs mégaoctets de mémoire contiguë en un seul appel à *AlloueMemExt*. De plus, le gestionnaire de mémoire avec les anciennes versions de RAMDRIVE.SYS ne peut traiter que 95 blocs de mémoire étendue, au moins un de ces blocs étant toujours utilisé comme un disque virtuel. Bien que TPMEMEXT vous permette d'allouer plus de 94 blocs (tant que RAMDRIVE.SYS n'est pas présent), vous ne pourrez pas en traiter plus si vous voulez que votre programme soit compatible avec RAMDRIVE.

Quatrièmement, la procédure *DeplaceMemExt*, utilisée pour transférer des données entre la mémoire normale et la mémoire étendue, ne permet pas de déplacer des données par octets mais seulement par mots (deux octets). Cela signifie que vous devez choisir avec une grande précaution vos structures de données et que vous devrez quelquefois compléter des types d'enregistrements, par exemple avec des octets fictifs pour garantir que leur taille soit un nombre pair. Le seul bénéfice de cette restriction est que vous pouvez déplacer jusqu'à 128 K de mémoire contiguë en un seul appel à *DeplaceMemExt*.

Cinquièmement, afin de s'adapter à RAMDRIVE.SYS, la plupart des routines de TPMEMEXT demanderont un peu plus d'1K sur la pile pour les variables locales. Ne l'oubliez pas lorsque vous utilisez l'unité dans des programmes qui travaillent avec la pile.

Une dernière limitation importante à ne pas oublier est que TPMEMEXT fonctionne actuellement avec DOS 3.x et 4.x. Cette restriction est destinée à éviter des conflits potentiels lors de l'exécution d'un programme sous OS/2 (lorsqu'il n'y aura plus de possibilité de conflit, cette restriction sera levée). Le fait que TPMEMEXT ne puisse être utilisée pour allouer de la mémoire sous DOS 2.x est sans signification puisque les machines 80286 et 80386 (les seules qui peuvent supporter la mémoire étendue) ne peuvent fonctionner sous DOS 2.x.

Pour un exemple d'utilisation des routines de TPEMS et TPMEMEXT, voir DIFF.PAS.

TPEMS : Gestion de la mémoire EMS

Cette section décrit les types, variables, constantes, procédures et fonctions se trouvant dans la section interface de l'unité TPEMS.

Constantes

```
CodeErreurEms = $0FFFF;
```

Utilisée par de nombreuses fonctions de TPEMS pour signaler une condition d'erreur.

AllouePagesEms

Déclaration

```
function AllouePagesEms(NumPages : Word) : Word;
```

Objet

Alloue le nombre de pages indiqué et renvoie un code d'identification.

Commentaires

Cette fonction permet d'allouer la mémoire étendue par blocs de 16 K (16384 octets) ou *pages*. Vous pouvez considérer qu'il s'agit d'une combinaison de *GetMem*, utilisée pour allouer la mémoire normale dans *Turbo Pascal*, et de la fonction d'ouverture de fichier du DOS qui renvoie un code d'identification de fichier. Comme pour les codes d'identification de fichiers DOS, le code d'identification renvoyé par *AllouePagesEms* est le moyen d'accéder à la mémoire que vous avez allouée. En particulier, il est nécessaire pour lier une page donnée à la mémoire normale (en utilisant *AssociePagesEms*) et pour libérer la mémoire lorsqu'elle n'est plus nécessaire (en utilisant *DeallouecodeIdentEms*).

AllouePagesEms renvoie *CodeErreurEms* en cas d'erreur. La cause d'erreur la plus courante est le manque de mémoire pour satisfaire la demande. Vous pouvez connaître le nombre de pages disponibles à n'importe quel moment en appelant *PagesEmsDisp*.

Remarque sur les codes d'identification : bien que la version standard du gestionnaire de mémoire EMS affecte des codes d'identifications dans l'ordre $FF00, $FE01, $FD02, etc, cet ordre n'est pas garanti pour toutes les versions de gestionnaire de mémoire EMS.

Exemple

```
PagesAAllouer := 4;
if (PagesEmsDisp >= PagesAAllouer) then begin
   MonGestEMS := AllouePagesEMS(PagesAAllouer);
   if (MonGestEMS = CodeErreurEms) then {Erreur};
end
else
   {Mémoire insuffisante};
```

Cet exemple montre comment allouer 64 K (4 pages de 16K) de mémoire EMS en utilisant *AllouePagesEms*. Remarquez que *PagesEmsDisp* est appelée tout d'abord pour s'assurer qu'il y a suffisamment de place en mémoire.

Voir aussi

DeallouecodeIdentEms (TPEMS)
AssociePageEms (TPEMS)
PagesEmsDisp (TPEMS)

AssociePageEms

Déclaration

```
function AssociePageEms(CodeIdent, PageLogique : Word;
                        PagePhysique : Byte) : Boolean;
```

Objet

Fait correspondre une *PageLogique,* associée à un code d'identification, à une *PagePhysique* (0-3).

Commentaires

Cette fonction est utilisée pour pouvoir accéder aux données enregistrées en mémoire EMS (*Remarque* : voir les explications de *PtrCadrePageEms* qui définissent certains termes utilisés ici). Cette fonction fait correspondre une *page logique* donnée (définie par les paramètres *CodeIdent* et *PageLogique*) avec l'une des quatres *pages physiques* de la structure de page du gestionnaire de mémoire EMS. C'est ce qui vous permet de traiter la mémoire EMS comme la mémoire normale.

Le paramètre *CodeIdent* est la valeur renvoyée précédemment par *AllouePagesEms* lorsque vous avez alloué votre mémoire. *PageLogique* est un numéro compris entre 0 et n-1 où n est le nombre de pages allouées à *CodeIdent* par *AllouePagesEms*. *PagePhysique* est un numéro compris entre 0 et 3 représentant l'une des quatres pages physiques qui composent la structure de pages.

AssociePageEms renvoie "True" si elle réussit et "False" si l'opération est interrompue. Les causes d'erreurs les plus vraisemblables sont un code d'identification non valide ou un numéro de page incorrect.

Exemple

```
type
   PageEms = array[1..16384] of byte;
   CadrePageEms = record
                     Page0, Page1, Page2 : PageEms;
                     Page3 : array[1..16369] of byte;
                  end;
   PtrCadrePage = ^CadrePageEms;
var
   MonPtrCadrePage : PtrCadrePage;
   MonPtrPageEms : ^PageEms;
   MonGestEms : word;
```

```
...
MonPtrCadrePage := PtrCadrePageEms;
if not AssociePageEMS(MonGestEms, 0, 2) then
   {erreur};
MonPtrPageEms := @MonPtrCadrePage^.Page2[1];
```

Cet exemple montre comment vous pourriez faire correspondre la première page logique (#0) associée à *MonCodeIdentEms* avec la troisième page physique (#2) et charge *MonPointeur* avec l'adresse de départ de la page physique.

Voir aussi

AllouePageEms (TPEMS)

PtrCadrePageEms (TPEMS)

DeallouеCodeIdentEms

Déclaration

```
function DeallouеCodeIdentEms(CodeIdent : Word) : Boolean;
```

Objet

Libère le code d'identification indiqué et la mémoire qui lui est associée.

Commentaires

Cette fonction EMS est équivalente à *Dispose* et *FreeMem*. Elle permet de libérer de la mémoire précédemment allouée avec *AllouePagesEms*. Remarquez que seul le code d'identification renvoyé par *AllouePagesEms* est nécessaire et non le nombre de pages qui lui est affecté. *DeallouеCodeIdentEms* renvoie "False" si le code d'identification ne peut être libéré. L'erreur la plus fréquente est un code d'identification non valide ou un code d'identification qui a déjà été libéré.

Très important : contrairement à TPMEMEXT, TPEMS ne libère pas automatiquement la mémoire EMS que vous avez allouée. Aussi, tout programme qui utilise TPEMS doit contenir *une routine* de sortie qui libèrera toute mémoire EMS allouée. Voir l'exemple qui suit.

Exemples

```
MonGestEMS := AllouePagesEMS(PagesAAllouer);
...
if not DeallouеCodeIdentEMS(MonGestEms) then
   {Erreur};
```

Cet exemple montre comment libérer la mémoire allouée préalablement avec *AllouePagesEms*.

```
{$F+}
procedure ProcSortie;
begin
   ExitProc := SauveProcSortie;
   if not DeallouеCodeIdentEMS(MonGestEms) then
      {erreur si la mémoire est déjà allouée};
end;
{$F-}
...
SauveProcSortie := ExitProc;
ExitProc := @ProcSortie;
```

Cet exemple met en oeuvre une routine de sortie qui libère toute la mémoire assignée par *MonCodeIdentEms* lorsque le programme se termine.

Voir aussi

AllouePagesEms (TPEMS)

EmsPresent

Déclaration

```
function EmsPresent : Boolean;
```

Objet

Renvoie "True" si le gestionnaire de mémoire EMS est présent.

Commentaires

Cette fonction indique si le gestionnaire de mémoire EMS (EMM) est installé. Elle n'indique pas si la carte mémoire correspondante aux spécifications EMS est installée. Vous devrez appeler cette routine avant d'utiliser n'importe quelle autre procédure et fonction de l'unité TPEMS. Dans de rares cas, vous pourriez avoir besoin d'appeler *VersionEms* pour être sûr qu'une fonction EMS particulière que votre programme utilise est disponible.

Exemple

```
if not EmsPresent then begin
   WriteLn('Ce programme nécessite une mémoire EMS. Abandon...');
   Halt(1);
end;
```

Dans cet exemple, un programme qui demande de la mémoire EMS pour lancer des messages d'erreurs si *EmsPresent* renvoie "False".

Voir aussi

VersionEms (TPEMS)

NbCodesIdentEmsActifs

Déclaration

```
function NbCodesIdentEmsActifs Word;
```

Objet

Renvoie le nombre de codes d'identification EMS actifs.

Commentaires

Cette fonction permet de déterminer le nombre de codes d'identification EMS actifs qui ont été assignés par le gestionnaire de mémoire EMS. Comme *TotalPagesEms*, elle est principalement utilisée pour l'affichage de statut. *NbCodesIdentEmsActifs* renvoie *CodeErreurEms* ($FFFF) en cas d'erreur. Les causes d'erreurs les plus vraisemblables sont les mêmes que celles de la fonction *StatutEmsOK*.

Exemple

```
WriteLn(LongInt(TotalPagesEms) * 16384, ' octets de mémoire EMS.');
WriteLn(LongInt(PagesEmsDisp) * 16384, ' octets disponibles.');
WriteLn(LongInt(NbCodesIdentEmsActifs), ' gestionnaires EMS actifs.');
```

Appel à *NbCodesIdentEmsActifs* qui permet au programme d'afficher le nombre de codes d'identification qui sont actifs, ainsi que d'autres d'informations.

Voir aussi

StatutEmsOK (TPEMS)

TotalPagesEms (TPEMS)

PagesEmsDisp

Déclaration

```
function PagesEmsDisp : Word;
```

Objet

Renvoie le nombre de pages disponibles avec le gestionnaire de mémoire EMS.

Commentaires

Cette fonction permet de savoir quelle est la mémoire EMS restante pour vos besoins. *PagesEmsDisp* renvoie le nombre de pages de 16 K (16384 octets) qui sont disponibles ou *CodeErreurEms* en cas d'erreur. Les causes d'erreurs les plus vraisemblables sont les mêmes que celles de la fonction *StatutEmsOK*. Pour savoir combien de pages sont réellement installées dans le système, vous pouvez utiliser *TotalPagesEms*.

Exemple

```
PagesAAllouer := PagesEmsDisp;
if (PagesAAllouer = CodeErreurEms) then
   {Erreur}
else
   if (PagesAAllouer <> 0) then
      GestEMS := AllouePagesEMS(PagesAAllouer);
```

Cet exemple montre comment allouer toute la mémoire EMS disponible à votre programme.

Voir aussi

StatutEmsOK (TPEMS)

TotalPagesEms (TPEMS)

PagesEmsPrises

Déclaration

```
function PagesEmsPrises(CodeIdent : Word) : Word;
```

Objet

Renvoie le nombre de pages prises par un code d'identification.

Commentaires

Cette fonction permet de déterminer le nombre de pages allouées pour un code d'identification donné. Elle renverra une valeur dans l'intervalle 1-512 ou *CodeErreurEms* ($FFFF) en cas d'erreur. La cause d'erreur la plus vraisemblable est un code d'identification non valide. *PagesEmsPrises* est utilisée principalement pour afficher des informations.

Exemple

```
var
   Gest, Pages : Word;
   EnregGest : record
                  OctetFort,
                  OctetFaible : byte;
               end absolute Gest;
begin
   Gest := $FF00;
   repeat
      Pages := PagesEmsPrises(Gest);
      if (Pages <> CodeErreurEms) then
         WriteLn('Gest $', MotHex(Gest), ': ', Pages, 'Pages');
      Inc(EnregGest.OctetFaible);
      Dec(EnregGest.OctetFort);
   until (Pages = CodeErreurEms);
end.
```

Cet exemple montre comment déterminer le nombre de pages alloué pour chaque code d'identification EMS actif. (Fonctionnement non garanti, voir les notes spéciales sur les codes d'identification dans les explications de *AllouePagesEms*).

Voir aussi

AllouePagesEms (TPEMS).

PtrCadrePageEms

Déclaration

```
function PtrCadrePageEms : Pointer;
```

Objet

Renvoie un pointeur sur l'adresse de base du cadre de page.

Commentaires

Cette fonction renvoie un pointeur sur l'adresse de base du *cadre de page* utilisé par le gestionnaire de mémoire EMS. Un cadre de page occupe un segment entier de 64 K divisé en quatre pages de 16 K numérotées de 0 à 3. Ces quatre pages sont appelées *pages physiques* en opposition aux *pages logiques* puisqu'il y a un mégaoctet de mémoire adressable dans le 808x. Les pages logiques au contraire existent sur les cartes mémoire EMS et leurs numéros sont des numéros relatifs par rapport à un code d'identification donné. Si vous allouez trois pages pour un code d'identification particulier avec *AllouePagesEms* par exemple, vous devrez les référencer individuellement comme les pages 0, 1 ou 2. Afin d'accéder au contenu d'une page logique donnée, vous devez attribuer le numéro de page dans l'une des quatres pages physiques en utilisant *AssociePagesEms*. Le meilleur moyen de comprendre le concept de cadre de page est de le voir comme une simple structure de données :

```
type
   PageEms = array[1..16384] of byte;
   CadrePageEms = record
                    Page0, Page1, Page2 : PageEms;
                    Page3 : array[1..16369] of byte;
                  end;
   PtrCadrePage = ^CadrePageEms;
```

Note : Les types de données en *Turbo Pascal* ne peuvent excéder 65521 octets. *Page3* est donc plus petite que les trois autres pages. Cependant, il sera possible de faire référence aux 16 384 éléments du tableau tant que le contrôle d'intervalle sera désactivé.

Pour déplacer 16 K de données dans la quatrième page physique (#2) par exemple, vous devriez écrire quelque chose comme :

```
MonPtrCadrePage := PtrCadrePageEms;
Move(Source, MonPtrCadrePage^.Page2[1], 16384);
```

Remarquez que TPEMS ne définit pas ces types pour vous car il n'est pas sûr que vous devrez traiter une page comme un tableau d'octets.

Exemple

```
var
   MonPtrCadrePage : PtrCadrePage;
   MonPtrPageEms : ^PageEms;
   MonGestEms    : Word;
...
MonPtrCadrePage := PtrCadrePageEms;
if not AssociePageEMS(MonGestEms, 0, 2) then {erreur};
MonPtrCadrePage := @MonPtrCadrePage^.Page2[1];
```

Cet exemple montre comment faire correspondre la première page logique (#0) associée à *MonGestEms* avec la troisième page physique (#2) et charger *MonPtrPageEms* avec l'adresse de départ de la page physique.

Voir aussi

AllouePagesEms (TPEMS)

AssociePageEms (TPEMS)

RestaureContexteEms

Déclaration

```
function RestaureContexteEms(CodeIdent : Word) : Boolean;
```

Objet

Restaure le contexte correspondant au gestionnaire de mémoire EMS pour le code d'identification spécifié.

Commentaires

Cette fonction est destinée à l'origine à une utilisation avec des "popups" qui utilisent de la mémoire EMS. En relation avec *SauveContexteEms*, elle permet de sauvegarder le contexte du gestionnaire de mémoire EMS en entrée de la "popup" et de le restaurer à la sortie, si bien que le programme suspendu n'est pas affecté. Le code d'identification passé en paramètre doit être le même que celui utilisé dans un appel précédent à *SauveContexteEms*. *RestaureContexteEms* renvoie "True" si l'opération se termine bien.

Exemple

```
{$F+}
procedure MaPopup;
begin
   if not SauveContexteEms(GestEMS) then
      Exit;
   {.. corps de la popup ..}
   if not RestaureContexteEms(GestEMS) then
      {Erreur. Affichage d'un avertissement};
end;
{$F- }
```

Cet exemple montre comment utiliser *RestaureContexteEms* dans une "popup". La "popup" devra sauvegarder le contexte en entrée et le restaurer à la sortie.

Voir aussi

SauveContexteEms (TPEMS)

SauveContexteEms

Déclaration

```
function SauveContexteEms(CodeIdent : Word) : Boolean;
```

Objet

Sauve le contexte de la mémoire EMS pour les programmes résidents.

Commentaires

Cette fonction est destinée à l'origine à être utilisée avec des "popups" qui utilisent la mémoire EMS. En relation avec *RestaureContexteEms,* elle permet de sauvegarder le contexte (par exemple, les registres de la page sur la carte mémoire) du gestionnaire de mémoire EMS en entrée de la "popup" et de le restaurer à la sortie. Le programme suspendu n'est pas affecté. Le code d'identification passé en paramètre doit avoir été alloué en appelant *AllouePageEms. SauveContexteEms* renvoie "True" si elle se termine bien.

Exemple

```
GestEMS := AllouePagesEMS(PagesAAllouer);
{ctrl de CodeErreurEms}
...
{$F+}
procedure MaPopup;
begin
   if not SauveContexteEms(GestEMS) then
      Exit;
   {.. corps de la popup ..}
   if not RestaureContexteEms(GestEMS) then
      {Erreur. Affichage d'un avertissement};
end;
{$F- }
```

Cet exemple montre comment utiliser *SauveContexteEms* dans une "popup". La "popup" devra sauvegarder le contexte en entrée et le restaurer à la sortie. *MonCodeIdentEms* est le code d'identification précédemment assigné au programme par l'appel initial à *AllouePageEms.*

Voir aussi

AllouePageEms (TPEMS) RestaureContexteEms (TPEMS)

StatutEmsOK

Déclaration

```
function StatutEmsOK : Boolean;
```

Objet

Renvoie "True" si le gestionnaire de mémoire EMS renvoie un status OK.

Commentaires

StatutEmsOK permet de vérifier le bon fonctionnement de la mémoire EMS. Elle renvoie "True" si le gestionnaire de mémoire EMS renvoie un statut OK et "False" sinon. Les causes d'erreurs les plus vraisemblables sont une erreur dans le code du gestionnaire de mémoire EMS ou un dysfonctionnement de la carte mémoire. Remarquez que *StatutEmsOK* doit être utilisée seulement après que *EmsPresent* ait été appelée pour déterminer si le gestionnaire de mémoire EMS est installé.

Exemple

```
if not EmsPresent then begin
   WriteLn('Ce programme nécessite une mémoire EMS. Abandon...');
   Halt(1);
end;
if not StatutEmsOk then begin
   WriteLn('Problème dans le soft ou dans le hard EMS. Abandon...');
   Halt(1);
end;
```

EmsPresent est utilisée pour vérifier que le gestionnaire de mémoire EMS est installé et *StatusEmsOK* est appelée pour vérifier qu'il fonctionne correctement.

Voir aussi

EmsPresent (TPEMS)

TotalPagesEms

Déclaration

```
function TotalPagesEms : Word;
```

Objet

Renvoie le nombre total de pages de mémoire EMS, y compris les pages allouées.

Commentaires

Cette fonction renvoie le nombre de pages de mémoire de 16 K installées ou *CodeErreurEms* en cas d'erreur. Les causes d'erreurs les plus vraisemblables sont les mêmes que celles de la fonction *StatutEmsOK*. Elle est utilisée principalement pour afficher des informations.

Exemple

```
WriteLn(LongInt(TotalPagesEms) * 16384, ' octets de mémoire EMS.');
```

Voir aussi

StatutEmsOK (TPEMS)

PagesEmsDisp (TPEMS)

VersionEms

Déclaration

```
function VersionEms : Byte;
```

Objet

Renvoie le numéro de version en code BCD du gestionnaire de mémoire EMS.

Commentaires

Cette fonction renvoie une valeur spécifiant le numéro de version du gestionnaire de mémoire EMS. La valeur est en code BCD (Binary Coded Decimal). Pour contrôler les versions 3.2 ou supérieures par exemple, vous pourriez utiliser 'If VersionEms >= $32 then...'. *VersionEms* renvoie 0 en cas d'erreur, ce qui indique normalement que le gestionnaire de mémoire EMS n'est pas installé. Remarquez que toutes les routines de l'unité TPEMS sont disponibles dans les versions 3.0 et supérieures du gestionnaire de mémoire EMS, aussi *VersionEms* devrait être rarement utilisée.

Exemple

```
if VersionEms < $32 then begin
   WriteLn('Ce programme nécessite un gestionnaire de mémoire étendu
            3.2 ou plus.');
   Halt(1);
end;
```

Voir aussi

EmsPresent (TPEMS)

TPMEMEXT : Gestion de la mémoire étendue

Cette section décrit tous les types, variables, constantes, procédures et fonctions de la section interface de l'unité TPMEMEXT.

Types

```
PtrEtendu = Pointer;
```

Un pointeur étendu contient une adresse de 24 bits. Il peut être converti en un entier long (et vice-versa) pour adresser la mémoire et vous pouvez vérifier si sa valeur est égale à Nil, comme vous le feriez pour un pointeur normal (qui contient une adresse segmentée).

AlloueMemExt

Déclaration

```
function AlloueMemExt(N : Word) : PtrEtendu;
```

Objet

Alloue N Kilo-octets de mémoire étendue.

Commentaires

Comme *GetMem, AlloueMemExt* permet d'allouer un bloc de mémoire étendue mais elle ne peut allouer la mémoire que par blocs de 1 K (1024 octets). Puisque la fonction renvoie Nil s'il n'y a pas suffisamment de place pour satisfaire la demande, il n'est pas nécessaire que vous appeliez *MemExtDisp* avant d'effectuer un appel à cette fonction. Un bloc de mémoire étendue allouée avec *AlloueMemExt* peut être libéré avec *DeallouememExt*.

Exemple

```
MonPtrEtendu := AlloueMemExt(KoAAllouer);
if MonPtrEtendu = nil then begin
   WriteLn('Impossible d''allouer la mémoire étendue');
   Exit
end;
```

Voir aussi

MemExtDisp (TPMEMEXT) DeallouememExt (TPMEMEXT)

DeallouеMemExt

Déclaration

```
function DeallouеMemExt(P : PtrEtendu) : Boolean;
```

Objet

Libère un bloc de mémoire étendue.

Commentaires

DeallouеMemExt libère un bloc de mémoire étendue allouée par *AllouеMemExt*. Elle fonctionne seulement si le bloc à libérer est le dernier bloc alloué. La fonction renvoie "False" s'il n'est pas possible de libérer la mémoire. Remarquez qu'il n'est pas nécessaire d'appeler *DeallouеMemExt* à la fin de votre programme puisque TPMEMEXT libèrera automatiquement toute mémoire allouée.

Exemple

```
MonPtrEtendu := AlloueMemExt(KoAAllouer);
...
if not DeallouеMemExt(MonPtrEtendu) then begin
   WriteLn('Impossible de libérer la mémoire allouée précédemment');
   Exit;
end;
```

AllouеMemExt est appelée pour allouer de la mémoire étendue, *DeallouеMemExt* pour libérer la mémoire allouée.

Voir aussi

AlloueMemExt (TPMEMEXT)

DeplaceMemExt

Déclaration

```
procedure DeplaceMemExt(Src, Dest : PtrEtendu; CompteurMot : Word);
```

Objet

Déplace *CompteurMot* mots dans la mémoire étendue de *Src* vers *Dest*.

Commentaires

Comme la procédure *Move* de Turbo, *DeplaceMemExt* permet de déplacer des blocs d'une partie de la mémoire dans une autre. Ceci étant, vous pouvez utiliser cette procédure pour transférer des données de la mémoire étendue dans la mémoire normale (ou vice et versa) ou entre deux parties de la mémoire étendue. Contrairement à *Move*, *DeplaceMemExt* ne permet pas de déplacer des données par blocs de un octet, une restriction très importante à garder à l'esprit lors de l'utilisation des routines de TPMEMEXT.

Remarque : Lorsque vous utilisez *Move* avec des pointeurs normaux, vous devez déréférencer les pointeurs :

```
Move(PointA^, PointB^, OctetsADeplacer);
```

En utilisant *DeplaceMemExt* avec des pointeurs de type *PtrEtendu*, toutefois, vous n'aurez pas à les déférencer (en utilisant le symbole '^'). Aussi, lors du transfert de données de la mémoire normale vers la mémoire étendue et vice-versa, l'adresse en mémoire normale doit tout d'abord être convertie d'une adresse segmentée (pointeur) en une adresse absolue (pointeur étendu) en utilisant *PtrEnEtendu*. Par exemple :

```
DeplaceMemExt(PtrEnEtendu(@MonTableau[1]), MonPtrEtendu, 512);
```

déplacera 512 mots (1024 octets) de *MonTableau[1]* vers l'adresse pointée par *MonPtrEtendu*.

Exemples

```
DeplaceMemExt(PtrEnEtendu(MonPtrBuff)^, MonPtrEtendu^, MotsADeplacer);
```

Incorrect. Le pointeur étendu ne doit pas être déférencé.

```
DeplaceMemExt(PtrEnEtendu(MonPtrBuff), MonPtrEtendu, MotsADeplacer);
```

Correct. Ceci déplacera *MotsADeplacer* mots de données de l'adresse en mémoire normale pointée par *MonPtrBuffer* vers l'adresse pointée par *MonPtrEtendu*.

Voir aussi

PtrEnEtendu (TPMEMEXT)

MarqueMemExt

Déclaration

```
procedure MarqueMemExt(var P : PtrEtendu);
```

Objet

Enregistre dans *P* l'emplacement du prochain bloc de mémoire étendue.

Commentaires

Comme la procédure *Mark* de Turbo, *MarqueMemExt* permet d'enregistrer l'emplacement du prochain bloc de mémoire disponible dans une variable. Vous pouvez alors allouer plusieurs blocs de mémoire étendue en utilisant *AlloueMemExt* et les libérer tous ensemble en une fois avec *RendMemExt*. Contrairement aux procédures *Mark*, *Release*, *GetMem* et *FreeMem* de Turbo, les procédures *MarqueMemExt* et *RendMemExt* de TPMEMEXT n'entrent pas en conflit avec *AlloueMemExt* et *DeallouemMemExt*. *P* devra être positionné à Nil s'il n'y a pas de mémoire étendue disponible.

Dans cet exemple, *MarqueMemExt* est appelée avant d'allouer plusieurs blocs de mémoire étendue avec *AlloueMemExt* et *RendMemExt* est appelée pour libérer la mémoire lorsqu'elle n'est plus nécessaire.

Exemple

```
MarqueMemExt(DebutMemExt);
PE1 := AlloueMemExt(KoAAllouer1);
PE2 := AlloueMemExt(KoAAllouer2);
PE3 := AlloueMemExt(KoAAllouer3);
...
RendMemExt(DebutMemExt);
```

Voir aussi

AlloueMemExt (TPMEMEXT)

RendMemExt (TPMEMEXT)

MemExtDisp

Déclaration

```
function MemExtDisp : Word;
```

Objet

Renvoie le nombre de Kilo-octets de la mémoire étendue qui sont actuellement disponibles.

Commentaires

Comme la fonction *MemAvail* de Turbo, *MemExtDisp* permet de déterminer la taille de la mémoire étendue disponible pour votre programme. Le résultat est fourni en Kilo-octets (1024 octets) puisque c'est l'unité utilisée par *AlloueMemExt* lors de l'allocation de mémoire étendue. La fonction renvoie zéro si la version DOS 3.x ou 4.x n'est pas utilisée, même s'il y a réellement de la mémoire étendue disponible.

Exemple

```
if MemExtDisp >= KoAAllouer then
   MonPtrEtendu := AlloueMemExt(KoAAllouer)
else begin
   WriteLn('Mémoire étendue insuffisante');
   MonPtrEtendu := Nil;
end;
```

Dans cet exemple, *MemExtDisp* est appelée pour vérifier s'il y a suffisamment de mémoire avant de faire une allocation avec *AlloueMemExt*.

Voir aussi

TotalMemExt (TPMEMEXT) AlloueMemExt (TPMEMEXT)

PtrEnEtendu

Déclaration

```
function PtrEnEtendu(P : Pointer) : PtrEtendu;
```

Objet

Convertit un pointeur normal en un pointeur étendu.

Commentaires

PtrEnEtendu convertit un pointeur normal représentant une adresse segmentée en un pointeur étendu représentant une adresse absolue. Entre autres choses, *PtrEnEtendu* est nécessaire pour déplacer des données entre la mémoire normale et la mémoire étendue en utilisant *DeplaceMemExt*. Remarquez que le pointeur passé à *PtrEnEtendu* doit appartenir à l'intervalle $0:$0 à $FFFF:$000F.

Exemple

```
DeplaceMemExt(PtrEnEtendu(MonPtrBuff), MonPtrEtendu, MotsADeplacer);
```

RendMemExt

Déclaration

```
procedure RendMemExt(P : PtrEtendu);
```

Objet

Libère toute la mémoire étendue au-dessus de *P*.

Commentaires

Comme *Release*, *RendMemExt* permet de libérer tous les blocs de mémoire au-dessus d'une adresse donnée, adresse enregistrée dans la variable de type *PtrEtendu* en utilisant *MarqueMemExt*. Vous pouvez allouer plusieurs blocs de mémoire étendue avec *AllouéMemExt* et les libérer tous en une seule fois avec *RendMemExt*. Contrairement aux procédures *Mark*, *Release*, *GetMem* et *FreeMem* de Turbo, les procédures *MarqueMemExt* et *RendMemExt* de TPMEMEXT n'entrent pas en conflit avec *AllouéMemExt* et *DeallouéMemExt*. Remarquez qu'il n'est pas nécessaire d'appeler *RendMemExt* à la fin de vos programmes puisque TPMEMEXT libérera automatiquement toute la mémoire allouée.

Très important : Le paramètre *P* doit avoir été initialisé avec *MarqueMemExt*. Ce ne peut pas être un pointeur étendu renvoyé par *AllouéMemExt*.

Exemple

```
MarqueMemExt(DebutMemExt);
PE1 := AllouéMemExt(KoAAllouer1);
PE2 := AllouéMemExt(KoAAllouer2);
...
RendMemExt(DebutMemExt);
```

MarqueMemExt est appelée avant d'allouer plusieurs blocs de mémoire étendue avec *AllouéMemExt* et *RendMemExt* les libère lorsque vous n'en avez plus besoin. L'appel à *RendMemExt* peut être inutile si le programme se termine.

Voir aussi

MarqueMemExt (TPMEMEXT)

TotalMemExt

Déclaration

```
function TotalMemExt : LongInt;
```

Objet

Renvoie le nombre total d'octets de la mémoire étendue du système.

Commentaires

Contrairement à *MemExtDisp* qui renvoie le nombre de Kilo-octets de la mémoire étendue réellement disponibles, *TotalMemExt* renvoie le nombre total d'octets de la mémoire étendue installée dans le système. Elle est utilisée à l'origine pour afficher des informations. La fonction renvoie zéro si la version DOS 3.x ou 4.x n'est pas utilisée, même si la mémoire étendue est réellement installée.

Exemple

```
WriteLn(MemExtDisp * 1024, ' octets de mémoire étendue disponibles sur ',
WriteLn(TotalMemExt);
```

Voir aussi

MemExtDisp (TPMEMEXT)

9 Macros

Généralités

Le terme *macro* est largement utilisé dans de nombreuses situations. De nombreux progiciels, notamment les tableurs, comportent un *macro langage* qui permet de personnaliser vos tâches. De la même façon, certains compilateurs et assembleurs permettent d'affecter un nom symbolique à une séquence d'instructions.

Les procédures "inline" de *Turbo Pascal* peuvent être considérées comme des macros dans ce sens. En général, le terme référence des macros entrées à partir du clavier, une facilité qui permet à partir d'une seule touche de faire un certain travail. La plupart des éditeurs et des traitements de texte comportent une telle facilité et *Turbo Professionnel* permet maintenant d'inclure cette caractéristique puissante dans vos propres applications.

L'unité TPMACRO contient toutes les routines de bas niveau dont vous aurez besoin pour cela. Elle permet de définir des macros, de leur affecter des noms, de les enregistrer, de les rejouer, de les sauvegarder dans un fichier pour une utilisation future et de les charger lorsque cela est nécessaire.

Une fois définies, les macros-clavier peuvent être rejouées de deux façons. Si l'exécution des macros est permise (par un appel à *MacrosOn*), une macro se rejouera automatiquement à n'importe quel moment quand la touche correspondante sera appuyée. Sinon, vous pouvez forcer l'exécution d'une macro donnée en appelant *LanceMacro* ou *LanceMacroParTouche*. Dans les deux cas, l'opération sera exécutée par le processeur de macros de TPMACRO qui peut également enregistrer des macros.

Une limitation importante mérite d'être notée ici. TPMACRO n'autorise pas les macros imbriquées. Par exemple, si une macro (appelons-la A) a été affectée à (F1), une autre macro (appelons-la B) a été affectée à (F2), et que la macro A contienne (F2), la macro B ne pourra être exécutée comme une partie de la macro A. Deux raisons principales à cette limitation. Tout d'abord cela permet d'éviter des boucles infinies, qui pourraient se produire avec une macro telle que A dans notre exemple qui contient (F2) et une macro

B qui contient (F1). Deuxièmement, elle permet d'utiliser TPMACRO pour reconfigurer le clavier (Exemple, vous pourriez l'utiliser pour inverser les fonctions de (F1) et (F3)).

Bien que cette possibilité d'enregistrer et de rejouer des macros-clavier soit suffisante pour certaines applications, dans de nombreux cas, vous aurez aussi besoin de modifier ces macros après les avoir définies. Pour cette raison, *Turbo Professionnel* contient une unité appropriée, TPEDMAC, qui implémente un éditeur de macros complet. Cet éditeur spécialisé permet facilement d'insérer, de supprimer, de modifier les touches d'une macro qui sont représentées comme des chaînes de 1 à plusieurs caractères. La touche Escape, par exemple, est affichée comme <Esc> et la touche Enter comme <Enter>. Aussi, vous n'aurez rien à connaître sur les codes de balayage afin de les utiliser.

Les macros-clavier de *Turbo Professionnel* sont enregistrées dans des enregistrements de taille variable, de type *EnrMacro*. La première rubrique de l'enregistrement *NbTouches* est un mot indiquant le nombre de touches dans la macro. La seconde rubrique, *TabTouches*, est un tableau de mots avec un code caractère dans l'octet de poids faible et le code de balayage dans l'octet de poids fort, le même format que celui utilisé par le BIOS. La fin du tableau est marquée par un mot avec la valeur de *FinMacro* ($FFFF ou -1).

Vous pourrez voir certaines similitudes entre des chaînes de Turbo qui commencent par un octet de longueur et les chaînes ASCIIZ qui utilisent le caractère nul (#0) pour marquer la fin des chaînes. Chaque macro est associée à une touche telle que (F1) et un nom facultatif tel que 'Aide'. Par défaut, vous êtes limité à 100 macros (*MaxMacros*) d'au plus mille touches (*MaxTouchesParMacro*) chacune. Mais comme *Turbo Professionnel* est livré avec un code source complet, vous pourrez augmenter ou diminuer ces limites quand vous le voulez.

Un exemple de démonstration d'utilisation des macros claviers de *Turbo Professionnel* et de l'éditeur de macros est le programme de démonstration SMACS.PAS. SMACS est un processeur de macros résident en mémoire centrale qui comporte la plupart des caractéristiques des programmes commercialisés tel que *SuperKey* de Borland. Il est livré avec un fichier de macros , TURBO5.MAC, que vous pouvez utiliser avec l'éditeur de *Turbo Pascal*. Pour plus de détails, voir les explications de SMACS au chapitre 13.

TPMACRO : Macros clavier

Cette section décrit tous les types, les variables, les constantes, les procédures et les fonctions qui se trouvent dans la partie interface de l'unité TPMACRO.

Constantes

```
FinMacro = $FFFF;
```

Utilisée pour marquer la fin d'une macro. Egalement utilisée pour marquer des entrées inutilisées dans le tableau *TouchesDéfinies*.

```
Int16H = 12;
```

Numéro du code d'identification de INT $16.

```
IDFichMacro : LongInt = $04001087;
```

Valeur identifiant un fichier de macros. Ces valeurs sont enregistrées au début de tout fichier de macros pour garantir sa validité.

```
MaxTouchesParMacro = 1000;
```

Nombre maximum de touches dans une macro. Si vous voulez augmenter cette limite, souvenez-vous que *EnrMacro* contient un array[1..MaxTouchesParMacro] de mots (voir ci-dessous), et que TPMACRO et TPEDMAC déclare une variable globale de type *EnrMacro*. Autrement dit, la valeur limite absolue de cette constante est légèrement supérieure à 32000, ou 16000 si TPEDMAC est utilisé. La valeur habituelle est beaucoup plus petite.

```
MaxMacros = 100;
```

Nombre maximum de macros que TPMACROS peut gérer à la fois.

```
PauseRejoue : Word = 0;
```

Nombre de millisecondes d'attente avant de rejouer une touche particulière dans une macro.

Types

```
EnrMacro = record
             NbTouches : Word;
             TabTouches : array[1..MaxTouchesParMacro] of Word;
             Depassement : Word;
           end;
```

Description d'une macro. *NbTouches* est le nombre de touches de la macro, non compris la dernière qui est toujours *FinMacro* ($0FFFF). *TabTouche* contient les touches réelles. *Dépassement* contient la valeur FinMacro lorsque *TabTouche* est entièrement rempli.

```
PtrEnrMacro = ^EnrMacro;
```

Pointeur sur un enregistrement macro.

```
PointeurChaine = ^String;
```

Pointeur sur une chaîne.

Variables

```
LongTouchesAux : array[0..255] of byte;
```

Longueur des noms de touches auxiliaires (celles avec les codes de balayages étendus).

```
TouchesDefinies : array[1..MaxMacros] of Word;
```

Table des touches macros définies. Les entrées inutilisées contiennent *FinMacro*.

```
MacroActive : Boolean;
```

Si égale à "True", une macro est en cours d'exécution.

```
NomMacro: array[1..MaxMacros] of Pointer;
```

Tableau des pointeurs sur les noms de macros.

```
PointeursMacros : array[1..MaxMacros] of Pointer;
```

Tableau des pointeurs sur les macros définies.

```
EnregistreMacro : Boolean;
```

Si égale à "True", l'enregistrement d'une macro est autorisé.

```
MacrosSontOn : Boolean;
```

Si égale à "True", l'exécution d'une macro est autorisée.

```
LongTouchesNormales : array[0..255] of Byte;
```

Longueur des noms des touches normales.

```
PartieMacro : EnrMacro;
```

Utilisée par le processeur de macros lors de l'enregistrement des touches clavier.

```
TouchePartieMacro : Word;
```

Peut être utilisée pour enregistrer la touche associée avec *PartieMacro*.

```
NomPartieMacro : string;
```

Peut être utilisée pour enregistrer le nom associé à *PartieMacro*.

AdresseMacro

Déclaration

```
function AdresseMacro(Touche : Word) : Pointer;
```

Objet

Renvoie un pointeur sur la macro pour une touche ou Nil si la touche n'est pas définie.

Commentaires

Cette routine permet de savoir si une touche donnée est associée ou non à une macro. Vous pourriez appeler cette routine, par exemple, avant d'appeler *LanceMacroParTouche* ou pour vérifier si vous devez questionner l'utilisateur pour savoir si une macro existante peut être écrasée.

Exemple

```
var
   ToucheSauvee : Word;
begin
   Write('Pressez la touche de la macro devant être rejouée : ');
   MacrosOff;
   ToucheSauvee := LitMotTouche;
   MacrosOn;
   WriteLn;
   if AdresseMacro(ToucheSauvee) = nil then begin
      WriteLn('Macro non définie');
      Exit;
   end;
   ...
   LanceMacroParTouche(ToucheSauvee);
end.
```

Voir aussi

LanceMacroParTouche (TPMACRO)

AffecteNomMacro

Déclaration

```
function AffecteNomMacro(Touche : Word; var Nom : string) : Boolean;
```

Objet

Affecte un nom à une macro.

Commentaires

Cette routine permet de modifier un nom de macro ou d'affecter un nom à une macro qui n'en a pas encore. Elle supprime le nom existant affecté à la macro s'il existe et renvoie "False" s'il n'y a pas assez de place pour *Nom* ou si la macro associée à *Touche* n'est pas trouvée.

Exemple

```
var
   Nom : String;
   TcheDeLaMacro : Word;
begin
   Write('Donnez le nom de la macro : ');
   ReadLn(Nom);
   if not AffecteNomMacro(TcheDeLaMacro, Nom) then
      WriteLn('Mémoire insuffisante ou macro introuvable.');
end.
```

Affecte un nouveau nom à la macro associée à *TcheDeLaMacro*.

Voir aussi

AlloueMacro (TPMACRO)

AlloueMacro

Déclaration

```
function AlloueMacro(Touche : Word; var Macro; var Nom : string):Byte;
```

Objet

Alloue une macro sur le tas.

Commentaires

Cette routine définit une macro et alloue de la place à la fois pour la macro et pour son nom (s'il existe). S'il y a déjà une macro affectée à *Touche*, l'ancienne macro et l'ancien nom sont supprimés et la place mémoire est libérée. *AlloueMacro* renvoie l'une des valeurs suivantes :

0 Macro allouée
1 Table macros pleine
2 Hors des limites du tas
3 Macro vide

Remarquez que la valeur 3 n'est pas nécessairement un code d'erreur.

Exemple

```
var
   ToucheMacroTemp : Word;
   MacroTemp : EnrMacro;
const
   ChaineVide : string[1] = '';
begin
   {..affectation de ToucheMacroTemp et de MacroTemp}
   case AlloueMacro(ToucheMacroTemp, MacroTemp, ChaineVide) of
   {les valeurs 1 et 2 sont des erreurs, 0 et 3 sont Ok}
      1 : WriteLn('Table des macros complète');
      2 : WriteLn('Hors mémoire');
   else
      WriteLn('Macro sauvegardée');
   end;
end.
```

Voir aussi

AffecteNomMacro (TPMACRO)
DeallouerMacro (TPMACRO)
DefinitMacro (TPMACRO)

CaracMacro

Déclaration

```
function CaracMacro(MacroCar : Char) : Word;
```

Objet

Convertit un caractère en une représentation code/caractère clavier.

Commentaires

Cette fonction permet de convertir facilement des caractères ASCII en codes touches. SMACS, par exemple, utilise cette routine dans son lecteur d'écran pour convertir des caractères pris sur l'écran en une macro.

Voir aussi

ToucheEnChaine (TPMACRO)

DeallouеMacro

Déclaration

```
function DeallouеMacro(Touche : Word) : Boolean;
```

Objet

Efface la macro associée à une touche et libère la mémoire correspondante.

Commentaires

Cette routine est la contre partie de *AllouеMacro*. Si une touche est associée à une macro, elle efface à la fois la macro et le nom correspondant (s'il existe) et libère leur mémoire. *DeallouеMacros* renvoie "False" si *Touche* n'est pas associée à une macro.

Exemple

```
if not DeallouеMacro(TouchePartieMacro) then {Error};
```

Se débarrasse de toute macro associée à *TouchePartieMacro*.

Voir aussi

AllouеMacro (TPMACRO)

DefinitMacro (TPMACRO)

DefinitMacro

Déclaration

```
function DefinitMacro(Touche : Word; var Macro) : Boolean;
```

Objet

Définit une nouvelle macro ou efface une macro existante.

Commentaires

Cette routine de bas niveau permet d'exécuter les fonctions principales de *AlloueMacro* et de *DeallouеMacro* sans allocation/déallocation automatique de mémoire dynamique. Elle est destinée à l'origine à des utilisations internes.

Pour définir une macro : *Touche* est la touche à associer à *Macro*. *DefinitMacro* retournera "False" si la table est pleine ou si la macro est vide.

Pour supprimer une macro : *Touche* est la touche à supprimer et *Macro* doit être une macro vide (*NbTouches* = 0). *DefinitMacro* renverra "False" si la macro n'est pas trouvée.

Voir aussi

AlloueMacro (TPMACRO) DeallouеMacro (TPMACRO)

EcritFichMacro

Déclaration

```
function EcritFichMacro(NomF : String) : Byte;
```

Objet

Ecrit un fichier de macros.

Commentaires

Cette routine enregistre les macros actuelles dans un fichier. Elle renvoie 0 en cas de succès, *IoResult* dans le cas contraire. *Remarque* : Puisque la routine ne contrôle pas les erreurs critiques du DOS, nous vous recommandons de ne pas l'utiliser dans des programmes résidents.

Exemple

```
if EcritFichMacro('MACROS.3') <> 0 then {};
```

Voir aussi

LitFichMacro (TPMACRO)

EffaceMacros

Déclaration

```
procedure EffaceMacros;
```

Objet

Efface toutes les macros actuelles.

Commentaires

EffaceMacros efface et supprime définitivement toutes les macros et les noms de macros.

Voir aussi

DeallouеMacro (TPMACRO) DefinitMacro (TPMACRO)

EnleveMacros

Déclaration

```
procedure EnleveMacros;
```

Objet

Restaure tous les vecteurs associés avec les macros.

Commentaires

Cette routine désactive le processeur de macros. Vous pourriez l'utiliser avant d'exécuter une commande du DOS par exemple. Le processeur de macros peut être réactivé en appelant *InitMacros*.

Exemple

```
EnleveMacros;
Code := ExecDos('', True, Nil);
if not InitMacros then {Erreur};
```

Voir aussi

InitMacros (TPMACRO)

EnregistrementMacroOff

Déclaration

```
procedure EnregistrementMacroOff;
```

Objet

Désactive l'enregistrement de macros.

Commentaires

L'appel de cette macro indique au processeur de macros d'arrêter l'enregistrement des touches dans *PartieMacro*. Il est de votre responsabilité d'affecter cette macro à une touche et d'allouer de la place pour elle (en utilisant *AlloueMacro* normalement).

Voir aussi

AlloueMacro (TPMACRO) EnregistrementMacroOn (TPMACRO)

EnregistrementMacroOn

Déclaration

```
procedure EnregistrementMacroOn;
```

Objet

Active l'enregistrement d'une macro.

Commentaires

L'appel de cette routine indique au processeur de macros de commencer l'enregistrement de toutes les touches entrées par l'utilisateur dans *PartieMacro*. Elle continuera d'enregistrer les touches jusqu'à ce que l'enregistrement soit désactivé par un appel à *EnregistrementMacroOff*.

Voir aussi

EnregistrementMacroOff (TPMACRO)

InitMacros

Déclaration

```
function InitMacros : Boolean;
```

Objet

Initialise le processeur de macros.

Commentaires

InitMacros active le processeur de macros. Vous pourriez le désactiver (en utilisant *EnleveMacros*) avant d'exécuter une commande du DOS par exemple, puis le réactiver avec *InitMacros*. Après un appel à *InitMacros*, aucune macro n'est définie et le traitement des macros et l'enregistrement des macros sont désactivés. *InitMacros* renvoie "False" si INT $16 ne peut être capturé. Ceci peut seulement arriver si votre programme a déjà utilisé le code d'identification ISR utilisé par le processeur de macros (Int16). Vous ne devriez pas avoir besoin d'appeler *InitMacros* au début de votre programme puisque cela est fait automatiquement.

Exemple

```
EnleveMacros;
Code := ExecDos('', True, Nil);
if not InitMacros then {Erreur};
```

Voir aussi

EnleveMacros (TPMACRO)

LanceMacro

Déclaration

```
procedure LanceMacro(Macro : Pointer);
```

Objet

Lance l'exécution d'une macro spécifiée.

Commentaires

Cette routine permet de lancer n'importe quelle macro, même si elle n'a pas été définie officiellement (avec *DefinitMacro* ou *AlloueMacro*). Si une macro est déjà en cours d'exécution, elle sera remplacée immédiatement par la macro passée en paramètre.

Exemples

```
var
   MonEnregMacro : EnrMacro;
   MonPtrMacro : PtrEnrMacro;
begin
   {..affectation de MonEnregMacro..}
   MonPtrMacro := @MonEnregMacro;
   LanceMacro(MonPtrMacro^);
end;
```

Incorrect. Le '^' dans MonPtrMacro^ n'est pas nécessaire.

```
LanceMacro(MonPtrMacro);
```

Correct.

```
LanceMacro(@MonEnregMacro);
```

Toujours correct.

Voir aussi

LanceMacroParTouche (TPMACRO)

LanceMacroParTouche

Déclaration

```
procedure LanceMacroParTouche(Touche : Word);
```

Objet

Lance l'exécution de la macro attachée à *Touche* si elle existe.

Commentaires

Contrairement à *LanceMacro*, cette routine permet de rejouer une macro lorsque vous connaissez simplement la touche qui lui est associée. Si une macro est déjà en cours d'exécution, elle sera immédiatement remplacée par la macro spécifiée.

Exemple

```
const
ToucheF1 = $3B00;
...
LanceMacroParTouche(ToucheF1);
```

Rejoue la macro associée à F1, si elle existe.

```
var
   Touche : Word;
begin
   Write('Pressez la touche de la macro devant être rejouée : ');
   MacrosOff;
   Touche := LitMotTouche;
   MacrosOn;
   ...
   LanceMacroParTouche(Touche);
end;
```

Lit une touche macro, effectue un traitement et rejoue la macro.

Voir aussi

LanceMacro (TPMACRO)

LitFichMacro

Déclaration

```
function LitFichMacro(NomF : String; Fusion : Boolean) : Byte;
```

Objet

Lit un fichier de macros.

Commentaires

Cette routine lit le contenu d'un fichier de macros en mémoire. Si *Fusion* est à "False", toute macro existante est supprimée avant que le fichier soit lu. Si *Fusion* est à "True", les macros existantes qui ne sont pas en contradiction avec les nouvelles sont laissées en mémoire. *LitFichMacro* renvoie 0 en cas de succès, $FF s'il n'y a pas de fichier macros valide, $FE en cas de mémoire insuffisante, sinon *IoResult*.

Remarque : Puisque cette routine ne contrôle pas les erreurs critiques du DOS, nous vous recommandons de ne pas l'utiliser dans des programmes résidents.

Exemples

```
if LitFichMacro('MACROS.1', False) <> 0 then {Erreur};
```

Les macros existantes sont écrasées.

```
if LitFichMacro('MACROS.2', True) <> 0 then {Erreur};
```

Les macros existantes ne sont pas écrasées.

Voir aussi

EcritFichMacro (TPMACRO)

MacrosOff

Déclaration

```
procedure MacrosOff;
```

Objet

Désactive l'exécution des macros.

Commentaires

Cette routine désactive l'exécution des macros. SMACS appelle cette routine, par exemple, avant de demander la touche associée à une macro à modifier. Une fois que l'utilisateur a appuyé sur une touche, l'exécution de la macro est réactivée par un appel à *MacrosOn*.

Exemple

```
MacrosOff;
W := LitMotTouche;
MacrosOn;
```

Désactive l'exécution des macros, lit au clavier et réactive l'exécution des macros ensuite.

Voir aussi

MacrosOn (TPMACRO)

MacrosOn

Déclaration

```
procedure MacrosOn;
```

Objet

Active l'exécution des macros.

Commentaires

Cette routine active l'exécution des macros. Certains programmes devront simplement appeler cette routine une première fois au début. Cependant, dans certains cas, vous devrez temporairement désactiver l'exécution d'une macro. SMACS, par exemple, désactive l'exécution des macros avant de demander la touche associée à une macro à modifier, puis la réactive en appelant cette routine.

Exemple

```
MacrosOff;
W := LitMotTouche;
MacrosOn;
```

Désactive l'exécution des macros, lit au clavier et réactive l'exécution des macros ensuite.

Voir aussi

MacrosOff (TPMACRO)

NbMacro

Déclaration

```
function NbMacro : Word;
```

Objet

Renvoie le nombre de macros qui ont été définies.

Commentaires

Le résultat est un nombre compris entre 0 et *MaxMacros* (256).

Exemple

```
WriteLn(NbMacro, ' macros définies');
```

RechercheIndexMacro

Déclaration

```
function RechIndexMacro(Touche : Word) : Word;
```

Objet

Recherche l'indice du tableau pour la macro affectée à *Touche*.

Commentaires

Cette routine renvoie l'indice du tableau pour la macro correspondant à *Touche* ou 0 si elle n'est pas trouvée. Cet indice peut être utilisé pour accéder aux éléments des tableaux *PointeursMacros*, *NomMacro* et *TouchesDefinies*. Elle est destinée à l'origine aux utilisations internes mais vous pouvez l'utiliser pour créer des routines qui ne peuvent être écrites en utilisant des routines de haut niveau de TPMACRO. Vous pourriez également l'utiliser comme une alternative rapide à *AdresseMacro* lorsque vous vérifiez si une touche particulière a déjà été définie. SMACS le fait en plusieurs endroits.

Exemple

```
Index := RechercheIndexMacro(TcheMacro);
if Index <> 0 then WriteLn('Nom : ', NomMacro[Index]^);
```

Voir aussi

AdresseMacro (TPMACRO)

ToucheEnChaine

Déclaration

```
procedure ToucheEnChaine(Touche : Word; var S : string;
                         var Special : Boolean);
```

Objet

Renvoie une chaîne (*S*) représentant une touche.

Commentaires

Cette routine peut être utilisée, par exemple, lorsque vous désirez afficher la liste des touches macros définies. *Special* est à "False" si un seul caractère (tel que **A**) est renvoyé et à "True" lorsque une touche avec un code de balayage étendu (tel que F1) est représentée.

Exemple

```
var
   I, Touche : Word;
   S : String;
   Special : boolean;
begin
   for I := 1 to MaxMacros do begin
      Touche := TouchesDefinies[I];
      if Touche <> FinMacro then begin
         ToucheEnChaine(Touche, S, Special);
         writeln(S);
      end;
   end;
end.
```

Affiche la liste des toutes les touches rattachées à des macros.

Voir aussi

CaracMacro (TPMACRO)

TPEDMAC : Editeur de macros

Cette partie décrit tous les types, les variables, les constantes, les procédures et les fonctions qui se trouvent dans la partie interface de l'unité TPEDMAC.

Constantes

```
NonToucheMacro = $FFFF;
```

Si cette valeur est passée à *ModifTouches* comme paramètre *Touche, Touche* n'est pas convertie en une chaîne de caractères et affichée pendant la modification.

Variables

```
MacroTemp : Enrmacro;
```

Macro temporaire utilisée pour la modification. Puisque cette macro est utilisée seulement pendant la modification, vous pouvez la traiter comme une macro temporaire dans la plupart des situations.

ModifTouches

Déclaration

```
function ModifTouches(Touche : Word; var Macro : Enrmacro;
                      ColMin, LigneMin, ColMax, LigneMax, : Byte;
                      Dim, Brillant : Byte;
                      var Modifiée : Boolean) : PtrEnrMacro;
```

Objet

Modifie la macro associée à *Touche*.

Commentaires

ModifTouches implémente un éditeur de macros complet. La fenêtre de modifications est définie par les paramètres *ColMin, LigneMin, ColMax, LigneMax*. La fenêtre devra avoir au moins une largeur de 40 colonnes et une hauteur de cinq lignes. Puisque la fenêtre se déroule verticalement, vous n'avez pas à vous inquiéter pour les macros qui sont trop longues pour être affichées en totalité.

Les paramètres *Dim* et *Brillant* définissent les attributs vidéo utilisés pour dessiner l'encadrement de fenêtre (*Brillant*) et les touches de la macro (les touches normales sont affichées avec l'attribut *Dim*, les touches qui doivent être représentées comme une chaîne de caractères telles que (ESC), (ENTER) ou (F1) sont affichées avec l'attribut *Brillant*.

Touche est la touche associée avec la macro et elle est normalement affichée dans le coin supérieur gauche de la fenêtre. Si vous ne voulez pas l'afficher, passez *NonToucheMacro* comme paramètre *Touche*.

Macro, une variable de type *EnrMacro*, n'est jamais réellement modifiée par *ModifTouches*. Au contraire, *ModifTouches* effectue une copie sur laquelle les modifications réelles sont exécutées et renvoie un pointeur sur la copie. Si aucune modification n'est faite sur la macro, le paramètre modifié est positionné à "False", sinon à "True".

Les commandes suivantes sont disponibles avec l'éditeur de macro :

(ENTER)	Arrête les modifications.
(ESC)	Restaure la macro d'origine et arrête la modification.
(←)	Curseur à gauche.
(→)	Curseur à droite.
(↑)	Ligne précédente.
(↓)	Ligne suivante.

(INS)	Bascule du mode insertion/recouvrement.
(DEL)	Efface la touche sous le curseur.
(BACKSPACE)	Efface la touche précédente.
(CTRL)-(BACKSPACE)	Efface toutes les touches de la macro.
(SCROLL LOCK)	Bascule entre le mode commande et le mode littéral. En mode littéral, en appuyant sur n'importe quelle touche listée ci-dessus, vous insérerez cette touche dans la macro sans exécuter la commande.

Exemple

```
const
   ChaineVide : string[1] = '';

var
   P, MonPtrMacro : PtrEnrMacro;
   Modifie : boolean;

begin
   MacrosOff;
   Write('Touche macro : ');
   TouchePartieMacro := LitMotTouche;
   MacrosOn;

   {la macro existe-t-elle ? }
   P := AdresseMacro(TouchePartieMacro);
   if P = Nil then begin
      {macro introuvable. Création d'une macro vide à éditer}
      PartieMacro.NbTouches := 0;
      PartieMacro.TabTouches[1] := FinMacro;
      P := @PartieMacro;
   end;

   {édition de la macro}
   MonPtrMacro := ModifTouches(TouchePartiemacro, P^, 1, 1,
            LargeurEcran, HauteurEcran, $07, $07, Modifie);
   ClrScr;

   {Si des modifications ont été apportées, il faut allouer la
    nouvelle macro}
   {l'ancienne sera écrasée}
   if Modifie then
      case AlloueMacro(TouchePartiemacro, MonPtrMacro^, ChaineVide) of
      1..2 : WriteLn('Hors mémoire');
   end;
end.
```

Cet exemple démontre comment mettre en oeuvre l'éditeur de macros en utilisant *ModifTouches*.

10 Grands tableaux

Généralités

Bien que la version 4.0 de *Turbo Pascal* fournisse le modèle "large" pour le code et ait toujours offert l'accès aux 640 K de la mémoire DOS via le tas, elle ne permet pas encore de traiter des tableaux individuels dépassant 64 KOctets. *Turbo Professionnel* fournit trois unités, TPTBMRAM, TPTBMVIR et TPTBMEXP -- pour remédier à cette restriction. *Turbo Professionnel* permet de traiter des grands tableaux à une ou deux dimensions et les éléments des tableaux peuvent être de n'importe quel type.

Les trois unités concernant les grands tableaux permettent de créer des tableaux de n'importe quelle taille. TPTBMRAM enregistre les tableaux de données dans la RAM normale et en conséquence est limitée aux tableaux qui tiennent exactement dans la partie disponible de la mémoire DOS de 640 K. TPTBMVIR est un moyen d'accès virtuel permettant l'enregistrement de la totalité d'un tableau dans un fichier en paginant des parties du tableau en mémoire à la demande. Les tableaux gérés par TPTBMVIR sont limités seulement par la taille de l'espace disque et la taille du fichier maximum autorisé par le système d'exploitation, en général 32 mégaoctets. TPTBMEXP enregistre les tableaux dans la mémoire EMS (Lotus/Intel/Microsoft). Ces tableaux sont limités par la version 3 EMS à 8 mégaoctets et plus récemment par la version 4 EMS à 32 mégaoctets.

L'interface d'appel pour ces trois unités est la même. Dans tous les cas, des noms de fonctions et procédures identiques sont utilisés pour accéder aux opérations fournies : allocation de tableaux, initialisation et déallocation ; positionnement, retour des éléments, enregistrement et chargement des tableaux sur et à partir du disque. Des différences mineures dans la liste des paramètres sont fonction de l'implémentation de chaque unité. Puisque les conventions d'appel sont identiques, les unités sur les tableaux peuvent être utilisées de façon interchangeable. L'utilisation de plus d'une unité dans un programme donné est également possible en tirant profit du mécanisme de masquage des noms d'unité de *Turbo Pascal 4*.

Par exemple :

```
TPTBMRAM.AlloueT(T, 100, 100, SIzeOf(Word));
TPTBMEXP.AlloueT(T, 100, 100, SIzeOf(Word));
TPTBMVIR.AlloueT(T, 100, 100, SIzeOf(Word), 'TABLEAU.DAT', 50000);
```

Chacune de ces unités est complètement dynamique, ce qui signifie que les dimensions et les types de données d'un tableau sont spécifiés à l'exécution plutôt qu'à la compilation. En retour, une seule copie du code des grands tableaux gérera plusieurs tableaux de données de tailles et de types différents dans un seul programme. L'inconvénient de cette approche est que l'unité de grands tableaux ne peut pas définir une fonction qui renvoie la valeur d'un élément de tableau, puisque le type de l'élément est inconnu jusqu'à l'exécution.

A la place de cela, chaque unité exporte une procédure qui renvoie les éléments de tableau comme des paramètres VAR sans type. Votre application peut facilement définir une fonction qui appelle cette procédure et renvoie un résultat de fonction de type approprié.

Les tableaux gérés par les unités de grands tableaux de *Turbo Professionnel* commencent toujours par l'indice zéro. L'indice minimum de chaque dimension d'un tableau est toujours zéro. Lorsqu'un nouveau tableau est alloué, vous pouvez spécifier le nombre total de lignes et de colonnes du tableau. Le plus grand indice du tableau sera égal au nombre réel de lignes ou de colonnes moins 1.

Les types de données dépassant 64 K octets sont plus longs à référencer que des petits tableaux avec l'architecture de la famille Intel 8086. En outre, les structures allouées dynamiquement sont plus longues d'accès que les structures adressées statiquement. Pour chaque accès à un élément de tableau, les routines de *Turbo Professionnel* doivent calculer une page (segment) et l'offset dans la page où la donnée sera réellement trouvée. Ce calcul est exécuté en assembleur pour obtenir la plus grande rapidité possible. Les tableaux virtuels EMS demandent une opération supplémentaire pour garantir que la page désirée est chargée en mémoire normale.

A titre indicatif, la table suivante fournit quelques résultats de temps de lecture et d'écriture d'éléments dans chaque modèle de tableau. Les performances de Turbo Pascal pour un petit tableau sont également montrées pour comparaison. Les mesures ont été prises sur un système 80286 à 8 mégaHertz. Les résultats en micro-secondes correspondent à la lecture et l'écriture d'un seul élément de tableau du type spécifié.

Elément	Turbo	RAM	Virtuel	EMS
Byte	8	88	125	135
Word	8	89	126	137
Real	16	93	129	141
String[80]	96	153	190	235

Les résultats du mode virtuel ne comportent pas les temps d'échange sur disque et les résultats du modèle EMS ne comportent pas les temps de recherche des pages. Les temps d'échange sur disque varient de façon significative en fonction du matériel. Les échanges de pages EMS prennent de 100 à 150 micro-secondes. TPTBMVIR supporte un nombre de pages en RAM défini par l'utilisateur et TPTBMEXP supporte deux structures de pages EMS indépendantes afin de minimiser le total des échanges.

Erreurs courantes

Pratiquement, tous les appels que nous recevons concernant ces trois unités sont le fruit d'une seule erreur : vous oubliez que les tableaux sont toujours indicés à partir de 0. Dans le cas le plus fréquent, l'utilisateur a déclaré un tableau à une dimension et essaie de référencer le nième élément comme la ligne n, la colonne 1, plutôt que comme la ligne n-1, colonne 0.

TPTBMRAM : Tableaux en RAM normale

Cette partie décrit tous les types, variables, constantes, procédures et fonctions qui se trouvent dans la section interface de l'unité TPTBMRAM.

Constantes

```
ErreurEcriture : Boolean = True;
ArretSurErreur : Boolean = True;
ControleIntervalle : Boolean = False;
```

Ces constantes typées déterminent la façon dont TPTBMRAM gère les erreurs. Si *ErreurEcriture* est à "True", un message d'erreur explicatif est écrit sur l'unité standard de sortie lorsqu'une erreur de tableau se produit. Si *ArretSurErreur* a la valeur "True", l'exécution est stoppée après que le message d'erreur ait été écrit. Si *ControleIntervalle* a la valeur "True", tous les accès aux éléments de tableau sont contrôlés et génèrent une erreur si les limites sont dépassées. Bien que le contrôle de bornes ralentisse les performances de TPTBMRAM, c'est la première chose à mettre en oeuvre en cas de mauvais fonctionnement. Toute application utilisant TPTBMRAM peut modifier ces constantes typées à l'exécution. Si *ArretSurErreur* vaut "False", l'application devra appeler la fonction *ErreurT* pour déterminer si une erreur de tableau s'est produite. Voir aussi les explications de *ErreurT*.

Types

```
TableauTp = Pointer;
```

Les tableaux en mémoire RAM utilisent seulement la place d'un pointeur dans le segment où ils sont déclarés. L'unité TPTBMRAM fonctionne de manière similaire à Modula-2 en exportant un type "flou" pour le tableau. En réalité, le type *TableauTp* pointe sur une structure d'enregistrement qui fournit d'autres informations sur les détails de l'implémentation. Un programme utilisant l'unité ne devrait pas accéder à ces détails pour lesquels il n'y a pas d'interface. Reportez-vous à la déclaration du type *TpArrayInternal* dans TPTBMRAM pour plus de détails.

```
TypeInit = (InitExact, InitRapide);
```

Ce type énuméré définit les options d'initialisation pour les tableaux. En passant la valeur *InitExact* à *InitT*, chaque élément du tableau sera initialisé à la valeur correspondante du type approprié. Avec *InitRapide*, chaque octet du tableau sera initialisé avec la valeur du premier octet de la valeur d'initialisation. Comme son nom l'indique, la deuxième option est plus rapide. *InitRapide* est pratique pour initialiser les tableaux numériques à 0 ou des tableaux de chaînes à la valeur chaîne vide (' ').

AlloueT

Déclaration

```
procedure AlloueT(var T : TableauTp; Lignes : Word; Cols : Word;
                  TailleElement : Word);
```

Objet

Alloue un tableau en mémoire RAM.

Commentaires

Appelez cette routine pour créer dynamiquement un nouveau tableau. Le paramètre *T* doit contenir une variable pointeur de type *TableauTp*. *Lignes* et *Cols* spécifient le nombre total d'éléments sur chaque dimension du tableau. *TailleElement* spécifie le nombre d'octets dans chaque élément du tableau. La meilleur façon de déterminer *TailleElement* est d'utiliser la fonction interne *SizeOf* de Turbo (voir exemple ci-dessous).

AlloueT définit automatiquement la structure de pages pour le tableau. Les critères de structure sont : chaque page est limitée à 64 K-octets, chaque page représente un sous-ensemble rectangulaire de la totalité du tableau et le temps de gestion des pointeurs des pages du tableau et des pages vides est minimisé. Lorsque *AlloueT* a défini une telle structure de page, elle appelle *PrimitiveAlloueT* qui alloue réellement l'espace nécessaire au tableau et initialise les rubriques d'en-tête nécessaires.

Remarquez que *AlloueT* n'effectue aucune initialisation des éléments du tableau. Sans initialisation ultérieure (peut-être en appelant *InitT*), les éléments du tableau contiendront des valeurs indéterminées. Si *AlloueT* trouve une erreur dans une spécification de tableau ou si la mémoire disponible n'est pas suffisante, elle renverra un code d'erreur dans la fonction *ErreurT* ou s'arrêtera en fonction des variables de contrôle des erreurs.

Exemples

```
AlloueT(T, 100, 100, SizeOf(Word));
```

Alloue un tableau de 100*100 de type Word.

```
AlloueT(T, 1000, 0, SizeOf(String));
```

Incorrect. Le nombre total de colonnes doit être égal au moins à 1.

```
AlloueT(T, 1000, 1, SizeOf(String));
```

Alloue un tableau uni-dimensionnel avec mille lignes, chacune pouvant contenir une chaîne de 255 caractères.

Voir aussi

ErreurT (TPTBMRAM)

PrimitiveAlloueT (TPTBMRAM)

ChargeT

Déclaration

```
procedure ChargeT(var T : TableauTp; NomFichier : string);
```

Objet

Alloue et lit un tableau à partir du disque.

Commentaires

ChargeT essaie tout d'abord d'ouvrir le fichier spécifié par *NomFichier*. Il est de la responsabilité du programme appelant de vérifier l'existence du fichier. Celui-ci doit d'abord avoir été créé par un appel à *EnregistreT*.

A partir du fichier, *ChargeT* lit un en-tête qui décrit la taille et le type du tableau. Avec les données, *ChargeT* appelle *PrimitiveAlloueT* pour allouer de la place sur le tas pour enregistrer le tableau, puis lit les données à partir du fichier et les place sur le tas et finalement ferme le fichier disque.

Remarquez que les formats de fichiers utilisés par TPTBMRAM, TPTBMVIR et TPTBMEXP ne sont pas interchangeables. Les routines *ChargeT* et *EnregistreT* sont des routines complémentaires dans chaque unité.

Exemple

```
ChargeT(T, 'STATS.TAB');
ChargeT(T, 'C:\DONNEES\101587.DON');
```

Voir aussi

ErreurT (TPTBMRAM)

ChargeTV

Déclaration

```
procedure ChargeTV(var T : TableauTP; NomFichier : string);
```

Objet

Alloue et lit un tableau à partir du disque en format virtuel.

Commentaires

Utilisez cette routine pour charger les tableaux précédemment initialisés par *EnregistreTV* ou par *EnregistreT* de l'unité TPTBMVIR. Le format de fichier TPTBMVIR peut être utilisé comme méthode d'échange de tableaux entre TPTBMRAM et TPTBMVIR. Cela peut être utile, lorsqu'un seul programme utilise TPTBMRAM ou TPTBMVIR, en fonction de la quantité de mémoire disponible détectée à l'exécution.

Avant d'appeler *ChargeTV*, vérifier que le nom du fichier existe.

Exemple

```
uses
   TpTbMRam;
var
   T : TableauTp;
begin
   ChargeTV(T, 'TEST.TAB');
end.
```

Charge un tableau précédemment enregistré par TPTBMRAM.EnregistreTV ou TPTBMVIR.EnregistreT.

Voir aussi

EnregistreT(TPTBMVIR) EnregistreTV(TPTBMRAM)

DealloueT

Déclaration

```
procedure DealloueT(var T : TableauTp);
```

Objet

Libère le tableau.

Commentaires

TPTBMRAM enregistre la totalité d'un grand tableau sur le tas. Toutes les fois que vous appelez *AlloueT*, le total de la mémoire nécessaire est alloué. En appelant *DealloueT*, vous libérerez tout l'espace alloué précédemment à un tableau.

Vous ne devez pas appeler *DealloueT* à la fin d'une application puisque la totalité du tas de Turbo sera libérée automatiquement à ce moment.

Exemple

```
AlloueT(T, 200, 200, SizeOf(Word));
DealloueT(T);
```

L'instruction *AlloueT* alloue approximativement 80000 octets sur le tas (200*200*2 plus environ 20 octets). *DealloueT* libère tout l'espace alloué par *AlloueT*.

Voir aussi

AlloueT (TPTBMRAM)

EcritElemT

Déclaration

```
procedure EcritElemT(T : TableauTp; L, C : Word; var Valeur);
```

Objet

Donne la valeur *Valeur* à l'élément de tableau T[L,C].

Commentaires

Tous les tableaux gérés par TPTBMRAM commencent à l'indice 0. Nous entendons par là que le plus petit index de chaque dimension du tableau est 0. De la même façon, le plus grand index de chaque dimension est égal au nombre total de lignes ou de colonnes du tableau moins 1.

EcritElemT fixe la valeur actuelle de l'élément à la position L,C. Vous pouvez définir le nom du tableau comme premier paramètre de *EcritElemT*. Puisque TPTBMRAM ne connaît pas le type de l'élément (seulement sa taille), la nouvelle valeur est passée comme un paramètre VAR sans type. Votre routine appelante devra passer une variable du type approprié dans cette position.

Si le contrôle d'intervalle est actif et que le numéro de ligne ou de colonne est en dehors des limites du tableau, *EcritElemT* s'arrêtera ou renverra une erreur via *ErreurT* en fonction de la valeur des variables de contrôle d'erreur.

Dans de nombreux cas, il est préférable de passer les nouveaux éléments de tableau en utilisant des constantes ou des expressions. TPTBMRAM ne peut pas supporter directement cela puisque les types de données du tableau ne sont pas connus à la compilation mais vous pouvez facilement définir une ou plusieurs procédures pour fixer les types de données spécifiques de votre application. Voici un exemple :

```
procedure EcritVal(T : TableauTp; L, C : Word; V : TypeUtl);
begin
   EcritElemT(T, L, C, V);
end;
```

EcritElemT peut alors être appelée avec des constantes ou des expressions dans la position *V*.

Remarquez que TPTBMRAM ne fournit aucun moyen de modifier une rubrique élémentaire dans un élément de tableau de type enregistrement. A nouveau, il est facile pour votre application de définir une telle routine :

```
procedure EcritChamp(T : TableauTp; L, C : Word; V : TypeChampUtl);
var
   X : TypeEnregUtl;
begin
   LitElemT(T, L, C, X);
   X.MonChamp := V;
   EcritElemT(T, L, C, X);
end;
```

Exemples

```
EcritElemT(T, 0, 0, W);
```

Donne au premier élément du tableau T la valeur de la variable W.

```
EcritElemT(T, TotalLignes-1, TotalColonnes - 1, W);
```

Donne au dernier élément du tableau T la valeur de W.

Voir aussi

ErreurT (TPTBMRAM)

LitElemT (TPTBMRAM)

EnregistreT

Déclaration

```
procedure EnregistreT(T : TableauTp; NomFichier : string);
```

Objet

Ecrit le tableau sur disque.

Commentaires

EnregistreT crée un nouveau fichier spécifié par le paramètre *NomFichier*. *EnregistreT* ne contrôle pas si ce fichier existe déjà, il est de la responsabilité du programme appelant de le faire.

EnregistreT écrit tout d'abord un en-tête sur le fichier, décrivant les dimensions et le type du tableau puis *EnregistreT* écrit chaque page du tableau sur disque et finalement ferme le fichier. *EnregistreT* ne libère pas l'image du tableau sur le tas. Si l'enregistrement d'un tableau est la dernière opération effectuée, appelez *DeallouetT* pour libérer la mémoire RAM une fois que le tableau a été écrit.

La taille du fichier permettant d'enregistrer le tableau ne sera que légèrement plus grande que la taille réelle donnée par *TotalLignes*TotalColonnes*TailleElement*.

EnregistreT renverra une erreur s'il n'est pas possible de créer, d'écrire ou de fermer le fichier du tableau.

Exemples

```
EnregistreT(T, 'C:\AUTOEXEC.BAT');
```

Prenez garde ! EnregistreT écrasera le fichier existant sans avertissement.

```
EnregistreT(T, 'STATS.DON');
```

Enregistre le tableau *T* dans STATS.DON.

Voir aussi

ChargeT (TPTBMRAM)

EnregistreTV

Déclaration

```
procedure EnregistreTV(T : TableauTP ; NomFichier : string );
```

Objet

Ecrit un tableau sur disque en format virtuel.

Commentaires

Utilisez cette routine pour enregistrer un tableau initialisé par TPTBMRAM dans un format qui puisse être lu et utilisé par TPTBMVIR. Cela est utile dans deux cas : un petit programme initialise des tableaux de constantes, utilisés ultérieurement par un gros programme (limité par la mémoire) qui doit par conséquent utiliser TPTBMVIR ; ou un seul programme peut appeler soit TPTBMRAM ou TPTBMVIR en fonction de la quantité de mémoire disponible détectée à l'exécution.

T est un grand tableau initialisé précédemment par des appels à TPTBMVIR. NomFichier est le nom du fichier où sera écrit le résultat. Si le fichier existe déjà, *EnregistreTV* le détruira sans avertissement.

Exemple

```
uses
   TpTbMRam;
const
   R : real = 0.0;
var
   T : TableauTp;
begin
   AlloueT(T, 100, 100, SizeOf(Real));
   InitT(T, R, InitRapide);
   EnregistreTV(T, 'TEST.TAB');
end.
```

Initialise un tableau de 100x100 réels avec des 0. Le fichier tableau TEST.TAB sera relu plus tard dans un programme, qui utilise TPTBMVIR avec l'instruction suivante :

```
ChargeT(T, 'TEST.TAB', OctetsRamPourDonnees');
```

Voir aussi

ChargeT(TPTBMVIR) ChargeTV(TPTBMRAM)

ErreurT

Déclaration

```
function ErreurT : Word;
```

Objet

Renvoie le dernier numéro d'erreur et réinitialise les variables de contrôle d'erreur.

Commentaires

Lorsque la variable de contrôle *ArretSurErreur* contient la valeur "False" (voir la partie déclaration), TPTBMRAM redonne le contrôle à l'application appelante après une erreur. Il est de la responsabilité de l'application d'appeler la fonction *ErreurT* pour déterminer si une erreur s'est produite ou non.

ErreurT fonctionne de la même manière que la fonction *IoResult* de *Turbo Pascal*. *ErreurT* renvoie une valeur 0 si l'opération précédente s'est bien terminée et une valeur spécifique différente de 0 si une erreur s'est produite. Après chaque appel à *ErreurT*, la variable de l'erreur interne est effacée.

Valeurs renvoyées par *ErreurT* :

1	Index de ligne hors des limites de l'intervalle (rapportée lors d'un appel à *LitElemT* ou *EcritElemT* seulement lorsque *ControleIntervalle* est à True)
2	Index de colonne hors des limites de l'intervalle (rapportée pendant un appel à *LitElemT* ou *EcritElemT* seulement si *ControleIntervalle* est à True)
3	Taille d'élément non valide (rapportée si *TailleElement* est à 0 lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
4	Limite de ligne ou de colonne non valide (rapportée si le nombre total de lignes ou de colonnes est égal à 0 lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
5	Taille de page non valide (rapportée si *LignesDansPage*ColsDansPage* dépasse 64K pendant un appel à *PrimitiveAlloueT*)
6	Taille d'index de page non valide (rapportée si *4*PagesLigne*PagesColonne* dépasse 64K lors d'un appel à *PrimitiveAlloueT*)

Valeurs renvoyées par *ErreurT* (suite)

7	Mémoire insuffisante pour allouer le tableau (rapportée lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
8	Erreur en création de fichier (rapportée lors d'un appel à *EnregistreT*)
9	Erreur d'écriture sur fichier (rapportée lors d'un appel à *EnregistreT*)
10	Erreur pendant la fermeture d'un fichier (rapportée lors d'un appel à *EnregistreT* ou *ChargeT*)
11	Fichier non trouvé (rapportée lors d'un appel à *ChargeT*)
12	Erreur de lecture sur fichier (rapportée lors d'un appel à *ChargeT*)
17	Format de fichier non valide (rapportée lors d'un appel à *ChargeT*)

Exemple

```
LitElemT(T, 20, 1000, V);
E := ErreurT;
if E <> 0 then begin
   Writeln('Erreur n° ', E, ' dans TPTBMRAM');
   Halt;
end;
```

Remarquez que E est d'abord enregistré dans une variable. Si *ErreurT* avait été appelée deux fois sur l'exemple, la deuxième valeur renvoyée aurait été 0.

Voir aussi

Déclarations (TPTBMRAM)

InitT

Déclaration

```
procedure InitT(T : TableauTp; var Valeur; Initialise : TypeInit);
```

Objet

Initialise tous les éléments du tableau avec *Valeur*.

Commentaires

Remarquez que *Valeur* est un paramètre VAR, aussi vous devrez affecter une valeur d'initialisation à la variable du type approprié et la passer à *InitT*. *Initialise* peut être soit *InitExact* ou *InitRapide* (voir *TypeInit*). Si *InitRapide* est utilisé, *InitT* initialise tout le tableau avec le premier octet de valeur. Si *InitExact* est utilisée, *InitT* affecte *Valeur* à chaque élément du tableau.

Exemples

```
InitT(T, 0, InitRapide);
```

Incorrect. Le paramètre *Valeur* doit être affecté à une variable.

```
Valeur := 0;
InitT(T, Valeur, InitRapide);
```

Utilisation correcte.

LitElemT

Déclaration

```
procedure LitElemT(T : TableauTp; L, C : Word; var Valeur);
```

Objet

Renvoie la valeur de T[L,C].

Commentaires

Tous les tableaux gérés par TPTBMRAM commencent à *l'indice 0*. Nous entendons par là que le plus petit index sur chaque dimension du tableau est égal à 0. Ceci entraîne que le plus grand index sur chaque dimension est égal au nombre total de lignes ou de colonnes dans le tableau moins 1.

LitElemT renvoie la valeur actuelle de l'élément de tableau situé à la position L,C. Vous pouvez spécifier le nom du tableau dans le premier paramètre de *LitElemT*. Puisque TPTBMRAM ne connaît pas le type de l'élément (seulement sa taille), elle renvoie la réponse dans un paramètre VAR non typé. Votre routine appelante devra passer une variable de type approprié à cette position.

Si le contrôle d'intervalle est actif et qu'une valeur de ligne ou de colonne est hors des limites du tableau, *LitElemT* s'arrêtera ou renverra une valeur d'erreur via *ErreurT* en fonction de la valeur des variables de contrôle d'erreur.

Dans de nombreux cas, il est pratique d'avoir une fonction (plutôt qu'une procédure) pour renvoyer les valeurs des éléments de tableau. L'unité TPTBMRAM ne peut pas définir une telle fonction puisque, à la compilation, le type des données du tableau n'est pas défini. Heureusement, il est très facile pour votre application de définir une ou plusieurs fonctions pour renvoyer les types de données spécifiques de votre application. Voici un exemple :

```
function LitT(T : TableauTp; L, C : Word) : TypeUtl;
var
   X : TypeUtl;
begin
   LitElemT(T, L, C, X);
   LitT := X;
end;
```

Ceci fonctionnera toutes les fois que *TypeUtl* est un type pour lequel les fonctions peuvent être définies. Ces types comprennent *Byte, Word, LongInt, Pointer, Real* et *String*. Malheureusement, les types enregistrements ne peuvent être renvoyés comme des résultats de fonctions.

Cependant, vous pouvez définir une fonction qui renvoie n'importe quelle rubrique élémentaire d'un enregistrement :

```
function LitChamp(T : TableauTp; L, C : Word) : TypeChampUtl;
var
     X : TypeEnregUtl;
begin
   LiElemT(T, L, C, X);
   LitChamp := X.MonChamp;
end;
```

Exemples

```
LitElemT(T, 0, 0, W);
```

Renvoie le premier élément du tableau *T* dans la variable *W*.

```
LitElemT(T, TotalLignes - 1, TotalColonnes - 1, W);
```

Renvoie le dernier élément du tableau *T*.

Voir aussi

ErreurT (TPTBMRAM) EcritElemT (TPTBMRAM)

PrimitiveAlloueT

Déclaration

```
procedure PrimitiveAlloueT(var T : TableauTp;
                           LignesDansPage, ColsDansPage : Word
                           PagesLigne, PagesColonne, TailleElement : Word);
```

Objet

Alloue un tableau en mémoire avec un contrôle complet de la structure de page.

Commentaires

PrimitiveAlloueT, comme son nom l'indique, est une alternative de bas niveau à la procédure *AlloueT*. Alors que *AlloueT* définit automatiquement la structure de page interne d'un tableau, *PrimitiveAlloueT* fournit au programme appelant un contrôle total sur la façon dont le tableau est organisé.

N'importe quelle structure de données en Turbo Pascal est limitée à 64 K-octets. Pour traiter de grands tableaux, TPTBMRAM divise une structure plus grande que 64 K en "pages" identiques, chacune d'elle étant plus petite que 64 K. TPTBMRAM gère ces pages en initialisant une liste de pointeurs, chacun d'entre eux pointant sur une page. Puis, toutes les fois qu'une référence de tableau est effectuée avec un numéro de ligne et de colonne donné, TPTBMRAM calcule le numéro de page et le déplacement dans la page où se trouve réellement la donnée.

AlloueT divise un tableau donné automatiquement en structures de pages, de façon optimum. Il est possible, toutefois, d'imaginer une structure de pages qui occupe moins de place sur le tas que *AlloueT*. Dans ce cas, vous devrez appeler *PrimitiveAlloueT* au lieu de *AlloueT*.

Par rapport aux paramètres passés à *PrimitiveAlloueT*, le nombre total de lignes dans le tableau est *PagesLigne*LignesDansPage* et le nombre total de colonnes est *PagesColonne*ColsDansPage*.

PrimitiveAlloueT renverra une erreur si elle détecte un problème avec la spécification de tableau ou s'il n'y a pas assez de place sur le tas.

Exemples

```
PrimitiveAlloueT(T, 200, 200, 1, 1, SizeOf(Word));
```

Incorrect. Ceci demande une taille de page de 80000 octets (200*200*2), ce qui dépasse la limite des 64 K.

```
PrimitiveAlloueT(T, 100, 100, 2, 2, SizeOf(Word));
```

Alloue un tableau de 200x200, de type Word.

```
PrimitiveAlloueT(T, 100, 0, 10, 1, SizeOf(String));
```

Incorrect. Le nombre total de colonnes doit être au moins égal à 1.

```
PrimitiveAlloueT(T, 100, 1, 10, 1, SizeOf(String));
```

Alloue un tableau uni-dimensionnel avec mille lignes, chacune pouvant contenir une chaîne de 255 caractères.

Voir aussi

ErreurT (TPTBMRAM)

AlloueT (TPTBMRAM)

TPTBMEXP : Tableaux en mémoire EMS

Cette partie décrit tous les types, variables, constantes, procédures et fonctions qui se trouvent dans la section interface de l'unité TPTBMEXP.

Constantes

```
ErreurEcriture: Boolean = True;
ArretSurErreur : Boolean = True;
ControleIntervalle : Boolean = False;
```

Ces constantes typées déterminent la façon dont TPTBMEXP gère les erreurs. Si *ErreurEcriture* a la valeur "True", un message d'erreur explicatif est écrit sur l'unité standard de sortie lorsqu'une erreur de tableau se produit. Si *ArretSurErreur* a la valeur "True", l'exécution est stoppée après que le message d'erreur ait été écrit. Si *ControleIntervalle* a la valeur "True", tous les accès aux éléments de tableau sont contrôlés et génèrent une erreur si les limites sont dépassées. Bien que le contrôle de bornes ralentisse les performances de TPTBMEXP, c'est la première chose à mettre en oeuvre en cas de mauvais fonctionnement. Toute application utilisant TPTBMEXP peut modifier ces constantes typées à l'exécution. Si *ArretSurErreur* a la valeur "False", l'application devra appeler la fonction *ErreurT* pour déterminer si une erreur de tableau s'est produite. Voir les explications de *ErreurT* pour plus de détails.

Types

```
TableauTp = Pointer;
```

Les tableaux en mémoire EMS utilisent seulement la place d'un pointeur dans le segment où ils sont déclarés. L'unité TPTBMEXP fonctionne de manière similaire à Modula-2 en exportant un type "flou" pour le tableau. En réalité, le type *TableauTp* pointe sur une structure d'enregistrement qui fournit d'autres informations sur les détails de l'implémentation. Un programme utilisant l'unité ne devrait pas accéder à ces détails pour lesquels il n'y a pas d'interface. Reportez-vous à la déclaration du type *TableauInterTp* dans la section implémentation de TPTBMEXP pour plus de détails.

```
TypeInit = (InitExact, InitRapide);
```

Ce type énuméré définit les options d'initialisation pour les tableaux. En passant la valeur *InitExact* à *InitT*, chaque élément du tableau sera initialisé à la valeur correspondante du type approprié. Avec *InitRapide*, chaque octet du tableau sera initialisé avec la valeur du premier octet de la valeur d'initialisation.

Comme son nom l'indique, la deuxième option est plus rapide. *InitRapide* est pratique pour initialiser les tableaux numériques à 0 ou des tableaux de chaînes à la valeur chaîne vide (' ').

Variables

```
EmsDisponible : Boolean ;
```

La partie initialisation de l'unité TPTBMEXP initialise cette variable à "True" ou "False" si elle détecte ou non un module pilote EMS. Si *EmsDisponible* vaut "False", votre application ne peut pas utiliser les routines de TPTBMEXP.

AlloueT

Déclaration

```
procedure AlloueT(var T : TableauTp; Lignes : Word; Cols : Word;
                  TailleElement : Word);
```

Objet

Alloue un tableau de la façon la plus simple.

Commentaires

Appelez cette routine pour créer dynamiquement un nouveau tableau. Le paramètre *T* doit contenir une variable de type *TableauTp*. *Lignes* et *Cols* spécifient le nombre total d'éléments sur chaque dimension du tableau. *TailleElement* spécifie le nombre d'octets dans chaque élément du tableau. La meilleure façon de définir *TailleElement* est d'utiliser la fonction interne de *Turbo Pascal SizeOf* (voir exemple ci-dessous).

La taille maximum de n'importe quel élément de tableau EMS est de 16 K octets. Cette limite est dûe à la taille du cadre de page égale à 16 K et le besoin d'au moins deux cadres indépendants en mémoire EMS.

La mémoire EMS impose une taille de page de 16 K octets pour les structures de tableau. Aussi l'utilisation réelle de la mémoire EMS sera toujours arrondie par excès à la valeur du prochain multiple de 16 K octets..

Remarquez que *AlloueT* n'effectue aucune initialisation des éléments du tableau. Sans initialisation ultérieure (peut-être en appelant *InitT*), les éléments du tableau contiendront des valeurs indéterminées.

Si *AlloueT* trouve une erreur dans une spécification de tableau ou si la mémoire disponible n'est pas suffisante, elle renverra un code d'erreur dans la fonction *ErreurT* ou s'arrêtera en fonction des variables de contrôle des erreurs.

Exemples

```
AlloueT(T, 100, 100, SizeOf(Word));
```

Alloue un tableau de 100*100 de type Word.

```
AlloueT(T, 1000, 0, SizeOf(String));
```

Incorrect. Le nombre total de colonnes doit être supérieur ou égal à 1.

```
AlloueT(T, 1000, 1, SizeOf(String));
```

Alloue un tableau uni-dimensionnel avec mille lignes, chacune pouvant contenir une chaîne de 255 caractères.

Voir aussi

ErreurT (TPTBMEXP)

ChargeT

Déclaration

```
procedure ChargeT(var T : TableauTp; NomFichier : string);
```

Objet

Alloue et lit un tableau à partir du disque.

Commentaires

ChargeT essaie tout d'abord d'ouvrir le fichier spécifié par *NomFichier*. Il est de la responsabilité du programme appelant de vérifier l'existence du fichier. Le fichier doit avoir d'abord été créé par un appel à *EnregistreT*.

A partir du fichier, *ChargeT* lit un en-tête qui décrit la taille et le type du tableau. Avec les données, *ChargeT* appelle *AlloueT* pour allouer de la mémoire pour enregistrer le tableau, puis lit les données à partir du fichier et les place dans l'espace alloué et finalement ferme le fichier.

Remarquez que les formats de fichiers utilisés par TPTBMRAM, TPTBMVIR et TPTBMEXP ne sont pas interchangeables. Les routines *ChargeT* et *EnregistreT* sont des routines complémentaires.

Exemple

```
ChargeT(T, 'STATS.TAB');
ChargeT(T, 'C:\DONNEES\101587.DON');
```

Voir aussi

ErreurT (TPTBMEXP)

DeallouеT

Déclaration

```
procedure DeallouеT(var T : TableauTp);
```

Objet

Libère le tableau.

Commentaires

Bien que TPTBMEXP range un petit tableau d'en-tête (environ 10 octets) sur le tas, la totalité d'un tableau EMS est enregistré en mémoire EMS. En appelant *DeallouеT*, vous libérerez à la fois le tas et la mémoire EMS occupée par le tableau.

Contrairement aux autres types de tableaux, il est indispensable que votre application appelle la routine *DeallouеT* pour chaque tableau alloué avant de retourner au DOS, sinon la mémoire EMS sera indisponible pour une autre utilisation.

Exemple

```
AlloueT(T, 200, 200, SizeOf(Word));
DeallouеT(T);
```

L'instruction *AlloueT* alloue approximativement 80000 octets sur le tas (200*200*2 plus environ 20 octets). *DeallouеT* libère tout l'espace alloué par *AlloueT*.

Voir aussi

AlloueT (TPTBMEXP)

EcritElemT

Déclaration

```
procedure EcritElemT(T : TableauTp; L,C : Word; var Valeur);
```

Objet

Place l'élément *Valeur* dans le tableau *T[L,C]*.

Commentaires

Tous les tableaux gérés par TPTBMEXP commencent à *l'indice 0*. Nous entendons par là que le plus petit index sur chaque dimension du tableau est égal à 0. De la même façon, le plus grand index de chaque dimension est égal au nombre total de lignes ou de colonnes du tableau moins 1.

EcritElemT fixe la valeur actuelle de l'élément de tableau à la position *L,C*. Puisque TPTBMEXP ne connaît pas le type de l'élément (seulement sa taille), la nouvelle valeur est passée comme un paramètre VAR sans type. Votre routine appelante devra passer une variable du type approprié dans cette position.

Si le contrôle d'intervalle est actif et que le numéro de ligne ou de colonne est en dehors des limites du tableau, *EcritElemT* s'arrêtera ou renverra une erreur via *ErreurT* en fonction du positionnement des variables de contrôle d'erreurs.

Dans de nombreux cas, il est préférable de passer les nouveaux éléments de tableau en utilisant des constantes ou des expressions. TPTBMEXP ne peut pas supporter directement cela puisque les types de données du tableau ne sont pas connus à la compilation, mais vous pouvez facilement définir une ou plusieurs procédures pour spécifier les types de données spécifiques de votre application. Voici un exemple :

```
procedure EcritVal(T : TableauTp; L, C : Word; V : TypeUtl);
begin
   EcritElemT(T, L, C, V);
end;
```

EcritElemT peut alors être appelée avec des constantes ou des expressions dans la position de *V*.

Remarquez que TPTBMEXP ne fournit aucun moyen de modifier une rubrique élémentaire dans un élément de tableau de type enregistrement. Voir l'entrée de *EcritElemT* dans TPTBMRAM pour trouver le moyen de contourner cette limitation.

EcritVal peut être appelée avec des constantes ou des expressions dans la position *V*.

Exemples

```
EcritElemT(T, 0, 0, W);
```

Donne au premier élément du tableau *T* la valeur de la variable *W*.

```
EcritElemT(T, TotalLignes - 1, TotalColonnes - 1, W);
```

Donne au dernier élément du tableau *T* la valeur de *W*.

Voir aussi

ErreurT (TPTBMEXP)

LitElemT (TPTBMEXP)

EnregistreT

Déclaration

```
procedure EnregistreT(T : TableauTp; NomFichier : string);
```

Objet

Ecrit le tableau sur disque.

Commentaires

EnregistreT crée un nouveau fichier spécifié par le paramètre *NomFichier*. *EnregistreT* ne contrôle pas si ce fichier existe déjà, il est de la responsabilité du programme appelant de le vérifier.

EnregistreT écrit tout d'abord un en-tête sur le fichier, décrivant les dimensions et le type du tableau puis écrit chaque page du tableau sur disque et finalement ferme le fichier. *EnregistreT* ne libère pas l'image du tableau sur le tas. Si l'enregistrement d'un tableau est la dernière opération effectuée, appelez *DealloueT* pour libérer la mémoire RAM, une fois que le tableau a été écrit.

La taille du fichier permettant d'enregistrer le tableau ne sera que légèrement plus grande que la taille réelle fournie par *TotalLignes*TotalColonnes*TailleElement*. Remarquez cependant, que les derniers 16 K du tableau sont écrits entièrement, indépendamment de la quantité réelle de données qu'il contient

EnregistreT renverra une erreur s'il n'est pas possible de créer, d'écrire ou de fermer le fichier du tableau.

Exemples

```
EnregistreT(T, 'C:\AUTOEXEC.BAT');
```

Prenez garde ! EnregistreT écrasera le fichier existant sans avertissement.

```
EnregistreT(T, 'STATS.DON');
```

Enregistre le tableau *T* dans STATS.DON.

Voir aussi

ChargeT (TPTBMEXP)

ErreurT

Déclaration

```
function ErreurT : Word;
```

Objet

Renvoie le dernier numéro d'erreur et réinitialise les variables de contrôle d'erreur.

Commentaires

Lorsque la variable de contrôle *ArretSurErreur* a la valeur "False" (voir la partie déclaration), TPTBMEXP redonne le contrôle à l'application appelante après une erreur. Il est de la responsabilité de l'application d'appeler la fonction *ErreurT* pour déterminer si une erreur s'est produite ou non.

ErreurT fonctionne de la même manière que la fonction *IoResult* de *Turbo Pascal*. *ErreurT* renvoie une valeur 0 si l'opération précédente s'est bien terminée et une valeur spécifique différente de 0 si une erreur s'est produite. Après chaque appel à *ErreurT*, la variable de stockage de l'erreur interne est effacée.

Valeurs renvoyées par *ErreurT* :

1	Index de ligne hors des limites de l'intervalle (rapportée lors d'un appel à *LitElemT* ou *EcritElemT* seulement lorsque *ControleIntervalle* est à "True")
2	Index de colonne hors des limites de l'intervalle (rapportée pendant un appel à *LitElemT* ou *EcritElemT* seulement si *ControleIntervalle* est à "True")
3	Taille d'élément non valide (rapportée si *TailleElement* est à 0 lors d'un appel à *AlloueT*)
4	Limite de ligne ou de colonne non valide (rapporté si le nombre total de lignes ou de colonnes est égal à 0 lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
7	Mémoire insuffisante pour allouer le tableau (rapportée s'il n'y a plus assez de place sur le tas lors d'un appel à *AlloueT*)
8	Erreur en création de fichier (rapportée lors d'un appel à *EnregistreT*)
9	Erreur d'écriture sur fichier (rapportée lors d'un appel à *EnregistreT*)

Valeurs renvoyées par *ErreurT* (suite)

10	Erreur pendant la fermeture d'un fichier (rapportée lors d'un appel à *EnregistreT* ou *ChargeT*)
11	Fichier non trouvé (rapporté lors d'un appel à *ChargeT*)
12	Erreur de lecture sur fichier (rapportée lors d'un appel à *ChargeT*)
13	Erreur d'association d'une page EMS (rapportée lors d'une tentative de réassociation de cadre de page manquant)
14	Module pilote EMS et/ou matériel non intallé (rapportée après une tentative d'utilisation de TPTBMEXP lorsque *EmsDisponible* vaut False)
15	Mémoire EMS insuffisante pour allouer le tableau (rapportée si la mémoire EMS est insuffisante lors d'un appel à *AlloueT*)
16	Erreur lors d'une libération de mémoire EMS (rapportée lors d'un appel à *DeallouetT*)
17	Format de fichier non valide (rapporté lors d'un appel à *ChargeT*)

Exemple

```
LitElemT(T, 20, 1000, V);
E := ErreurT;
if E <> 0 then begin
   Writeln('Erreur n° ', E, ' dans TPTBMEXP');
   Halt;
end;
```

Remarquez que *E* est d'abord enregistré dans une variable. Si *ErreurT* avait été appelé deux fois dans l'exemple, la deuxième valeur renvoyée aurait été égale à 0.

Voir aussi

Déclarations (TPTBMEXP)

InitT

Déclaration

```
procedure InitT(T : TableauTp; var Valeur; Initialise : TypeInit);
```

Objet

Initialise tous les éléments du tableau avec *Valeur*.

Commentaires

Valeur est un paramètre VAR, aussi vous devrez affecter une valeur d'initialisation à la variable du type approprié et la passer à *InitT*. *Initialise* peut être *InitExact* ou *InitRapide* (voir *TypeInit*). Si *InitRapide* est utilisé, *InitT* initialise tout le tableau avec le premier octet de Valeur. Si *InitExact* est utilisée, *InitT* affecte *Valeur* à chaque élément du tableau.

Exemples

```
InitT(T, 0, InitRapide);
```

Incorrect. Le paramètre *Valeur* doit être affecté auparavant à une variable.

```
Valeur := 0;
InitT(T, Valeur, InitRapide);
```

Utilisation correcte.

LitElemT

Déclaration

```
procedure LitElemT(T : TableauTp; L, C : Word; var Valeur);
```

Objet

Renvoie la valeur de T[L,C].

Commentaires

Tous les tableaux gérés par TPTBMEXP commencent à *l'indice 0*. Nous entendons par là que le plus petit index sur chaque dimension du tableau est égal à 0. Ceci entraîne que le plus grand index sur chaque dimension est égal au nombre total de lignes ou de colonnes dans le tableau moins 1.

LitElemT renvoie la valeur actuelle de l'élément de tableau situé à la position *L,C*. Puisque TPTBMEXP ne connaît pas le type de l'élément (seulement sa taille), *LitElemT* renvoie la réponse dans un paramètre VAR non typé. Votre routine appelante devra passer une variable de type approprié à cette position.

Si le contrôle d'intervalle est actif et qu'une valeur de ligne ou de colonne est hors des limites du tableau, *LitElemT* s'arrêtera ou renverra une valeur d'erreur via *ErreurT* en fonction de la valeur des variables de contrôle d'erreur.

Dans de nombreux cas, il est préférable d'avoir une fonction (plutôt qu'une procédure) pour renvoyer les valeurs des éléments de tableau. L'unité TPTBMEXP ne peut pas définir une telle fonction puisque, à la compilation, le type des données du tableau n'est pas défini. Heureusement, il est très facile pour votre application de définir une ou plusieurs fonctions pour renvoyer les types de données spécifiques de votre application. Voici un exemple :

```
function LitT(T : TableauTp; L, C : Word) : TypeUtl;
var
   X : TypeUtl;
begin
   LitElemT(T, L, C, X);
   LitT := X;
end;
```

Ceci fonctionnera toutes les fois que *TypeUtl* est un type pour lequel les fonctions peuvent être définies. Ces types comprennent *Byte, Word, LongInt, Pointer, Real* et *String*. Malheureusement, les types enregistrements ne peuvent être renvoyés comme des résultats de fonctions. Voir l'entrée de *LitElemT* dans TPTBMRAM pour trouver le moyen de contourner cette limitation.

Exemples

```
LitElemT(T, 0, 0, W);
```

Renvoie le premier élément d'un tableau *T* dans la variable *W*.

```
LitElemT(T, TotalLignes - 1, TotalColonnes - 1, W);
```

Renvoie le dernier élément du tableau *T*.

Voir aussi

ErreurT (TPTBMEXP)

EcritElemT (TPTBMEXP)

TPTBMVIR : Tableaux en mémoire virtuelle

Cette partie décrit tous les types, variables, constantes, procédures et fonctions qui se trouvent dans la section interface de l'unité TPTBMVIR.

Constantes

```
ErreurEcriture: Boolean = True;
ArretSurErreur : Boolean = True;
ControleIntervalle : Boolean = False;
```

Ces constantes typées déterminent la façon dont TPTBMVIR gère les erreurs. Si *ErreurEcriture* a la valeur "True", un message d'erreur explicatif est écrit sur l'unité standard de sortie lorsqu'une erreur de tableau se produit. Si *ArretSurErreur* a la valeur "True", l'exécution est stoppée après que le message d'erreur ait été écrit. Si *ControleIntervalle* a la valeur "True", tous les accès aux éléments de tableau sont contrôlés et génèrent une erreur si les limites sont dépassées. Bien que le contrôle de bornes ralentisse les performances de TPTBMVIR , c'est la première chose à mettre en oeuvre en cas de mauvais fonctionnement. Toute application utilisant TPTBMVIR peut modifier ces constantes typées à l'exécution. Si *ArretSurErreur* a la valeur "False", l'application devra appeler la fonction *ErreurT* pour déterminer si une erreur de tableau s'est produite. Voir les explications de *ErreurT* pour plus de détails.

```
EcritDisque : Boolean = False ;
```

EcritDisque définit la fréquence des écritures physiques sur disque lorsque la totalité d'un tableau virtuel est enregistrée. Si *EcritDisque* vaut "False", les pages sont écrites seulement lorsqu'elles sont supprimées de la mémoire. La page index n'est pas écrite et les buffers DOS ne sont pas vidés tant que le tableau n'est pas fermé ou que la commande *ApureT* n'est pas exécutée. Si *EcritDisque* vaut "True", la page index est mise à jour toutes les fois que l'image RAM de l'index est modifiée et les buffers DOS sont vidés après chaque écriture sur disque. Le positionnement de *EcritDisque* à "True" peut entrainer un ralentissement significatif de la pagination. Nous recommandons d'utiliser *ApureT* conjointement à *EcritDisque* = "*False*" lors de l'écriture des programmes.

Types

```
TableauTp = Pointer;
```

Les tableaux en mémoire RAM utilisent seulement la place d'un pointeur dans le segment où ils sont déclarés. L'unité TPTBMVIR fonctionne de manière similaire à Modula-2 en exportant un type "flou" pour le tableau. En réalité, le type *TableauTp* pointe sur une structure d'enregistrement qui fournit d'autres informations sur les détails de l'implémentation. Un programme utilisant l'unité ne doit pas accéder à ces détails qui ne sont donc pas interfacés. Reportez-vous à la déclaration du type *TpArrayInternal* dans la section implémentation de TPTBMVIR pour plus de détails.

```
TypeInit = (InitExact, InitRapide);
```

Ce type énuméré définit les options d'initialisation pour les tableaux. En passant la valeur *InitExact* à *InitT*, chaque élément du tableau sera initialisé à la valeur correspondante du type approprié. Avec *InitRapide*, chaque octet du tableau sera initialisé avec la valeur du premier octet de la valeur d'initialisation. Comme son nom l'indique, la deuxième option est plus rapide. *InitRapide* est pratique pour initialiser les tableaux numériques à 0 ou des tableaux de chaînes à la valeur chaîne vide (' ').

AlloueT

Déclaration

```
procedure AlloueT(var T : TableauTp;  Lignes : Word; Cols : Word;
                  TailleElement : Word; NomFich : String;
                  RamDonnees : LongInt);
```

Objet

Alloue un tableau virtuel.

Commentaires

Appelez cette routine pour créer dynamiquement un nouveau tableau. Le paramètre *T* doit contenir une variable pointeur de type *TableauTp*. *Lignes* et *Cols* spécifient le nombre total d'éléments sur chaque dimension du tableau. *TailleElement* spécifie le nombre d'octets dans chaque élément du tableau. La meilleur façon de définir *TailleElement* est d'utiliser la fonction interne de *Turbo Pascal SizeOf* (voir exemple ci-dessous).

NomFich spécifie le chemin du fichier qui contiendra le tableau virtuel. Il doit y avoir assez de place sur le disque pour contenir la taille réelle du tableau. *AlloueT* ne contrôle pas l'existence d'un fichier de même nom. S'il en existe un, il sera écrasé.

RamDonnees spécifie la taille d'une mémoire RAM nécessaire à la pagination. Plus vous allouerez de place en mémoire RAM, moins il y aura de pages manquantes et plus les performances seront augmentées. Vous devez allouer assez de mémoire RAM pour contenir au moins une page de tableau. Vous pouvez allouer de la mémoire pour contenir le tableau tout entier, auquel cas il n'y aura pas d'accès disque sauf pour effectuer les sauvegardes.

AlloueT définit automatiquement la structure de pages pour le tableau. Les critères de structure sont : chaque page est limitée à 64 K-octets, chaque page représente un sous-ensemble rectangulaire de la totalité du tableau et le temps de gestion des pointeurs des pages du tableau et des pages vides est minimisé. Ceci conduit *AlloueT* à minimiser le nombre de pages en taillant chaque page au plus près de 64 K octets. Après avoir défini une telle structure de page, *AlloueT* appelle *PrimitiveAlloueT* qui alloue réellement la place du tableau et initialise les rubriques de l'entête. Si *AlloueT* partage le tableau en *TotalPages* et si *PagesRam* sont résidentes à tout moment, l'occupation mémoire et l'espace disque peuvent être calculés comme suit :

```
RAMDeService = (6 * TotalPages) + (4 * PagesRam) + 200 octets
EspaceRamDonnées = (TaillePage + 20) * PagesRam octets
DisqueService = (4 * TotalPages) + 20 octets
EspaceDisqueDonnees = TaillePage * TotalPages
```

Remarquez que *AlloueT* n'effectue aucune initialisation des éléments du tableau. Sans initialisation ultérieure (peut-être en appelant *InitT*), les éléments du tableau contiendront des valeurs indéterminées.

Si *AlloueT* trouve une erreur dans une spécification de tableau ou si la mémoire disponible n'est pas suffisante, elle renverra un code erreur dans la fonction *ErreurT* ou s'arrêtera en fonction des variables de contrôles des erreurs.

Exemples

```
AlloueT(T, 1000, 0, SizeOf(String), 'GRAND.TAB', 100000);
```

Incorrect. Le nombre total de colonnes doit être supérieur ou égal à 1.

```
AlloueT(T, 1000, 1, SizeOf(String), 'GRAND.TAB', 100000);
```

Alloue un tableau mono-dimensionnel avec mille lignes, chacune capable de contenir une chaîne de 255 caractères.

Voir aussi

ErreurT (TPTBMVIR)

PrimitiveAlloueT (TPTBMVIR)

ApureT

Déclaration

```
procedure ApureT(T : Tableau Tp) ;
```

Objet

Mise à jour du fichier tableau sur le disque

Commentaires

ApureT trancrit les pages RAM qui sont différentes de celles du fichier disque et met à jour la page *Index* (qui décrit l'offset de chaque page). Finalement, elle vide les buffers DOS pour garantir que le fichier physique est totalement mis à jour. Le fichier tableau virtuel reste ouvert.

Exemple

```
ApureT(T);
```

Met à jour le fichier tableau *T* sur le disque

ChargeT

Déclaration

```
procedure ChargeT(var T : TableauTp; NomFichier : string;
                  RamDonnees : LongInt);
```

Objet

Alloue et lit un tableau à partir du disque.

Commentaires

ChargeT essaie tout d'abord d'ouvrir *NomFich*. Le programmeur doit s'assurer que le fichier existe. Le fichier est en format binaire. Il doit avoir été créé par un appel à *EnregistreT*.

ChargeT lit l'entête du fichier qui décrit la taille et le type du tableau. A partir de ces données, *ChargeT* appelle *PrimitiveAlloueT* pour allouer de la mémoire sur le tas. Puis *ChargeT* lit l'index de la page sur le disque et la place sur le tas. Le fichier disque reste ouvert pour les accès suivants aux données virtuelles. Les données ne sont lues en mémoire que lors de la pagination. *RamDonnees* est la taille de la mémoire à réserver sur le tas pour les pages de données. Elle doit pouvoir contenir au moins une page et n'est pas nécessairement la même que celle spécifiée à la création du tableau. En plus des pages de données, *ChargeT* a besoin de place supplémentaire sur le tas pour stocker les indices qui adressent le tableau : environ (*6 * TotalPagesTableau*) + 200 octets.

Considérons par exemple un tableau de 300 K octets, comportant 5 pages de 60 K octets chacune ; il faudra environ 230 octets sur le tas, plus environ 60 K octets pour chaque page de tableau résidente.

Remarquez que les formats de fichiers utilisés par TPTBMRAM, TPTBMVIN et TPTBMEXP ne sont pas interchangeables. Les routines *ChargeT* et *EnregistreT* sont des routines complémentaires.

Exemples

```
ChargeT(T, 'STATS.TAB', MemAvail div 2);
```

Alloue la moitié de la mémoire disponible sur le tas pour le stockage des données du tableau STATS.TAB.

```
ChargeT(T, 'C:\DONNEES\101587.DON', 1000);
```

Faites attention ! Si la taille de la page de ce tableau dépasse 1000 octets (ce qui arrive très souvent), cette instruction provoquera une erreur.

Voir aussi

ErreurT (TPTBMVIR)
AlloueT (TPTBMVIR)
PrimitiveAlloueT (TPTBMVIR)
EnregistreT (TPTBMVIR)

DealloueT

Déclaration

```
procedure DealloueT(var T : TableauTp; EffaceFich : Boolean);
```

Objet

Libère la mémoire occupée par le tableau et efface le fichier correspondant si *EffaceFich* vaut "True".

Commentaires

TPTBMVIR enregistre la totalité d'un grand tableau sur le tas. Toutes les fois que vous appelez *AlloueT* ou *PrimitiveAlloueT*, le total de la mémoire nécessaire est alloué. En appelant *DealloueT*, vous libérerez tout l'espace alloué précédemment à un tableau. Si le paramètre *EffaceFich* vaut "True", le fichier utilisé pour enregistrer le tableau virtuel est supprimé. S'il vaut "False", le fichier est mis à jour et fermé. Vous n'avez pas à appeler *DealloueT* à la fin d'une application puisque Turbo libère automatiquement le tas tout entier à ce moment.

Exemple

```
AlloueT(T, 200, 200, SizeOf(Word), 'TEST.TAB', 80000);
DealloueT(T, True);
```

L'instruction *AlloueT* alloue approximativement 80000 octets sur le tas (200*200*2 plus environ 20 octets). *DealloueT* libère les 80 000 octets et détruit TEST.ARR.

Voir aussi

AlloueT (TPTBMVIR)

EcritElemT

Déclaration

```
procedure EcritElemT(T : TableauTp; L,C : Word; var Valeur);
```

Objet

Donne à l'élément de tableau *T[L,C]* la valeur *Valeur*.

Commentaires

Tous les tableaux gérés par TPTBMVIR commencent à *l'indice 0*. Nous entendons par là que le plus petit index sur chaque dimension du tableau est égal à 0. De la même façon, le plus grand index de chaque dimension est égal au nombre total de lignes ou de colonnes du tableau moins 1.

EcritElemT fixe la valeur actuelle de l'élément de tableau à la position L,C. Vous spécifiez le nom de la variable tableau comme premier paramètre de *EcritElemT*. Puisque TPTBMVIR ne connaît pas le type de l'élément (seulement sa taille), la nouvelle valeur est passée comme un paramètre VAR sans type. Votre routine appelante devra passer une variable du type approprié dans cette position.

Si le contrôle d'intervalle est actif et que le numéro de ligne ou de colonne est en dehors des limites du tableau, *EcritElemT* s'arrêtera ou renverra une erreur via *ErreurT* selon la valeur des variables de contrôle d'erreur.

Dans de nombreux cas, il est préférable de passer les nouveaux éléments de tableau en utilisant des constantes ou des expressions.

TPTBMVIR ne peut pas gérer cela directement puisque les types de données du tableau ne sont pas connus à la compilation, mais vous pouvez facilement définir une ou plusieurs procédures pour spécifier les types de données spécifiques de votre application.

Voici un exemple :

```
procedure EcritVal(T : TableauTp; L, C : Word; V : TypeUtl);
begin
   EcritElemT(T, L, C, V);
end;
```

EcritVal peut alors être appelée avec des constantes ou des expressions dans la position *V*. Remarquez que TPTBMVIR ne fournit aucun moyen de modifier une rubrique élémentaire dans un élément de tableau de type enregistrement. Voir l'entrée de *LitElemT* pour le moyen de contourner cette limitation.

Exemples

```
EcritElemT(T, 0, 0, W);
```

Donne au premier élément du tableau *T* la valeur de la variable *W*.

```
EcritElemT(T, TotalLignes - 1, TotalColonnes - 1, W);
```

Initialise la valeur du dernier élément du tableau *T*.

Voir aussi

ErreurT (TPTBMVIR)

LitElemT (TPTBMVIR)

EnregistreT

Déclaration

```
procedure EnregistreT(T : TableauTp);
```

Objet

Ecrit le tableau sur disque et ferme le fichier .

Commentaires

Pour les tableaux virtuels, *EnregistreT* exécute la même chose que *ApureT*, mais ferme en plus le fichier tableau. Puisque la plus grande partie d'un tableau virtuel est déjà sur fichier, aucun nom de fichier n'est nécessaire.

La taille du fichier disque peut être calculée comme expliquée dans *AlloueT*.

EnregistreT renverra une erreur s'il n'est pas possible d'écrire sur le fichier ou de le fermer.

Un tableau enregistré n'est pas utilisable sauf s'il est rechargé en mémoire. Normalement, chaque appel à *EnregistreT* devrait être suivi d'un appel à *DeallouetT*.

Exemple

```
EnregistreT(T);
```

Voir aussi

DeallouetT (TPTBMVIR)

AlloueT (TPTBMVIR)

ErreurT

Déclaration

```
function ErreurT : Word;
```

Objet

Renvoie le dernier numéro d'erreur et réinitialise les variables de contrôle d'erreur.

Commentaires

Lorsque la variable de contrôle *ArretSurErreur* a la valeur "False" (voir la partie déclaration), TPTBMVIR redonnera le contrôle à l'application appelante après une erreur. Il est de la responsabilité de l'application d'appeler la fonction *ErreurT* pour déterminer si une erreur s'est produite ou non.

ErreurT fonctionne de la même manière que la fonction *IoResult* de *Turbo Pascal*. *ErreurT* renvoie une valeur 0 si l'opération précédente s'est bien terminée et une valeur spécifique différente de 0 si une erreur s'est produite. Après chaque appel à *ErreurT*, la variable de stockage de l'erreur interne est effacée.

Valeurs renvoyées par *ErreurT* :

1	Index de ligne hors des limites de l'intervalle (rapportée lors d'un appel à *LitElemT* ou *EcritElemT* seulement lorsque *ControleIntervalle* est à True)
2	Index de colonne hors des limites de l'intervalle (rapportée pendant un appel à *LitElemT* ou *EcritElemT* seulement si *ControleIntervalle* est à True)
3	Taille d'élément non valide (rapportée si *TailleElement* est à 0 lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
4	Limite de ligne ou de colonne non valide (rapportée si le nombre total de lignes ou de colonnes est égal à 0 lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
5	Taille de page non valide (rapportée si *LignesDansPage*ColsDansPage* dépasse 64K pendant un appel à *PrimitiveAlloueT*)
6	Taille d'index de page non valide (rapportée si *4*PagesLigne*PagesColonne* dépasse 64K lors d'un appel à *PrimitiveAlloueT*)

Valeurs renvoyées par *ErreurT* (suite):

7	Mémoire insuffisante pour allouer le tableau (rapportée lors d'un appel à *AlloueT* ou *PrimitiveAlloueT*)
8	Erreur en création de fichier (rapportée lors d'un appel à *EnregistreT*, *AlloueT* ou *PrimitiveAlloueT*)
9	Erreur d'écriture sur fichier (rapportée lors d'un appel à *EnregistreT*, ou d'une pagination)
10	Erreur pendant la fermeture d'un fichier (rapportée lors d'un appel à *EnregistreT*, *ChargeT*, ou *DeallouetT*)
11	Fichier non trouvé (rapportée lors d'un appel à *ChargeT*)
12	Erreur de lecture sur fichier (rapportée lors d'un appel à *ChargeT*)
13	Nombre de pages RAM <1 (rapportée lors d'un appel à *AlloueT*, *PrimitiveAlloueT* ou *ChargeT* si l'espace disponible sur le tas ne peut pas contenir au moins une page de tableau)
17	Format de fichier non valide (rapportée lors d'un appel à *ChargeT*)

Exemple

```
LitElemT(T, 20, 1000, V);
E := ErreurT;
if E <> 0 then begin
   Writeln('Erreur n° ', E, ' dans TPTBMVIR');
   Halt;
end;
```

Remarquez que *E* est d'abord enregistré dans une variable. Si *ErreurT* avait été appelé deux fois dans l'exemple, la deuxième valeur renvoyée aurait été 0.

Voir aussi

Déclarations (TPTBMVIR)

InitT

Déclaration

```
procedure InitT(T : TableauTp; var Valeur; Initialise : TypeInit);
```

Objet

Initialise tous les éléments du tableau avec *Valeur*.

Commentaires

Remarquez que *Valeur* est un paramètre VAR, aussi vous devrez affecter une valeur d'initialisation à la variable du type approprié et la passer à *InitT*. *Initialise* peut être *InitExact* ou *InitRapide* (voir *TypeInit*). Si *InitRapide* est utilisé, *InitT* initialise tout le tableau avec le premier octet de valeur. Si *InitExact* est utilisé, *InitT* affecte *Valeur* à chaque élément du tableau.

Exemples

```
InitT(T, 0, InitRapide);
```

Incorrect. Le paramètre *Valeur* doit être affecté à une variable.

```
Valeur := 0;
InitT(T, Valeur, InitRapide);
```

Utilisation correcte.

LitElemT

Déclaration

```
procedure LitElemT(T : TableauTp; L, C : Word; var Valeur);
```

Objet

Renvoie la valeur de *T[L,C]*.

Commentaires

Tous les tableaux gérés par TPTBMVIR commencent à *l'indice 0*. Nous entendons par là que le plus petit index sur chaque dimension du tableau est égal à 0. Ceci entraîne que le plus grand index sur chaque dimension est égal au nombre total de lignes ou de colonnes dans le tableau moins 1.

LitElemT renvoie la valeur actuelle de l'élément de tableau situé à la position L,C. Vous spécifiez le nom de la variable tableau dans le premier paramètre de *LitElemT*. Puisque TPTBMVIR ne connaît pas le type de l'élément (seulement sa taille), il renvoie la réponse dans un paramètre VAR non typé. Votre routine appelante devra passer une variable de type approprié à cette position.

Si le contrôle d'intervalle est actif et qu'une valeur de ligne ou de colonne soit hors des limites du tableau, *LitElemT* s'arrêtera ou renverra une valeur d'erreur via *ErreurT* en fonction de la valeur des variables de contrôle d'erreur.

Dans de nombreux cas, il est préférable d'avoir une fonction (plutôt qu'une procédure) pour renvoyer les valeurs des éléments de tableau. L'unité TPTBMVIR ne peut pas définir une telle fonction puisque, à la compilation, le type des données du tableau n'est pas défini. Heureusement, il est très facile pour votre application de définir une ou plusieurs fonctions pour renvoyer les types de données spécifiques de votre application. Voici un exemple :

```
function LitT(T : TableauTp; L, C : Word) : TypeUtl;
var
   X : TypeUtl;
begin
   LitElemT(T, L, C, X);
   LitT := X;
end;
```

Ceci fonctionnera toutes les fois que *TypeUtl* est un type pour lequel les fonctions peuvent être définies. Ces types comprennent *Byte, Word, LongInt, Pointer, Real* et *String*. Malheureusement, les types enregistrements ne peuvent être renvoyés comme des résultats de fonctions. Cependant, vous pouvez définir une fonction qui renvoie n'importe quelle rubrique élémentaire d'un enregistrement.

Exemples

```
LitElemT(T, 0, 0, W);
```

Renvoie le premier élément d'un tableau *T* dans la variable *W*.

```
LitElemT(T, TotalLignes - 1, TotalColonnes - 1, W);
```

Renvoie le dernier élément du tableau *T*.

Voir aussi

ErreurT (TPTBMVIR) EcritElemT (TPTBMVIR)

PrimitiveAlloueT

Déclaration

```
procedure PrimitiveAlloueT(var T : TableauTp;
                           LignesDansPage, ColsDansPage,
                           PagesLigne, PagesColonne, TailleElement :Word;
                           NomFich : String ; RamDonnees : LongInt) ;
```

Objet

Alloue un tableau en mémoire avec un contrôle total de la structure de page.

Commentaires

PrimitiveAllouePage, comme son nom l'indique, est une alternative de bas niveau à la procédure *AlloueT*. Alors que *AlloueT* définit automatiquement la structure de page interne d'un tableau, *PrimitiveAlloueT* fournit au programme appelant un contrôle total sur la façon dont le tableau est organisé.

AlloueT partage automatiquement et de façon optimale un tableau donné en structures de page. Dans le cas des tableaux virtuels, cependant, *AlloueT* perd une donnée importante qui pouvait lui permettre d'optimiser les performances ainsi que l'occupation mémoire. Puisque la vitesse de pagination sur disque est beaucoup plus lente que la vitesse CPU ou la vitesse des accès mémoire, le nombre d'accès disque devra être minimisé pour maintenir des performances acceptables sur le tableau. Si un tableau est adressé d'abord sur les lignes (le numéro de ligne est incrémenté rapidement et le numéro de colonnes moins souvent), chaque page devra contenir le nombre maximum de lignes. L'inverse pour un adressage par colonne. *AlloueT* pourrait choisir une mauvaise organisation interne.

Par rapport aux paramètres passés à *PrimitiveAlloueT*, le nombre total de lignes dans le tableau est *PagesLigne*LignesDansPage* et le nombre de colonnes est *PagesColonne*ColsDansPage*. Le nombre total de pages est *PagesLigne*PageColonne* et la taille de chaque page est *LignesDansPage*ColonnesDansPage*TailleElement*. Voir *AlloueT* pour déterminer les besoins des services d'un tableau et pour plus de détails sur *NomFich* et *RamDonnees*.

PrimitiveAlloueT renverra une erreur si elle détecte un problème avec la spécification de tableau ou s'il n'y a pas assez de place sur le tas.

Exemples

```
PrimitiveAlloueT(T, 200, 200, 1, 1, SizeOf(Word), 'DONNEES.TAB',
                 100000);
```

Incorrect. Ceci demande une taille de page de 80000 octets (200*200*2), ce qui dépasse la limite des 64 K.

```
PrimitiveAlloueT(T, 100, 100, 2, 2, SizeOf(Word), 'DONNEES.TAB',
                 100000);
```

Alloue un tableau de 200x200, de type Word. Si 100 000 octets sont disponibles sur le tas, le tableau tout entier sera résident et il n'y aura pas de pagination.

```
PrimitiveAlloueT(T, 100, 1, 10, 1, SizeOf(String), 'DONNEES.TAB',
                 100000);
```

Alloue un tableau mono-dimensionnel avec mille lignes, chacune capable de contenir une chaîne de 255 caractères. 3 pages sur 10 seront résidentes.

Voir aussi

ErreurT (TPTBMVIR) AlloueT (TPTBMVIR)

11 Arithmétique BCD

Généralités

Quelquefois, une très grande précision est demandée pour des calculs avec des nombres réels. Des exemples évidents portent sur les calculs financiers où des sommes de 1 milliard de francs doivent contenir des centimes ou des tables d'amortissements avec une précision de dix chiffres sur trente années.

Le type de réel de *Turbo Pascal* sur six octets ne fournit pas une précision suffisante dans ces cas-là. Deux problèmes se posent : tout d'abord, six octets ne permettent pas de garantir une précision suffisante et, deuxièmement, l'enregistrement des données en format binaire entraîne des erreurs d'arrondi lors des conversions.

La version 3.0 de *Turbo Pascal* fournissait une version spéciale du compilateur pour résoudre ces problèmes. Ce compilateur définissait le type réel avec un format BCD de dix octets (Binary Coded Decimal) et fournissait une précision de dix huit chiffres dans l'intervalle 1E-63 à 1E+63.

Les versions 4.0 et 5.0 de *Turbo Pascal* ne permettent plus l'arithmétique BCD. Bien que les types de données 8087 supportant au plus dix neuf chiffres significatifs soient disponibles, ces réels exigent la présence d'un co-processeur arithmétique et utilisent encore un format binaire provoquant des erreurs d'arrondi.

L'unité TPBCD de *Turbo Professionnel* offre un ensemble de routines arithmétiques BCD. Un réel BCD est représenté comme un tableau [1..10] d'octets, avec le même format interne que les réels BCD de *Turbo Pascal* 3.0. A côté de cet ensemble complet de routines de conversions de formats et de comparaisons, TPBCD fournit un ensemble de fonctions transcendantes qui n'existaient pas précédemment : *Sqrt, Exp, Ln, Sin, Cos* et *ArcTan*. De plus, TPBCD comporte une fonction *Formate* qui permet facilement de formater des résultats numériques.

Les routines de TPBCD sont codées en langage assembleur pour obtenir une meilleure performance et un code mémoire minimum. Si les routines transcendantes ne sont pas utilisées, TPBCD ajoutera environ 3K à la taille d'un programme. Si n'importe quelle routine transcendante est appelée, 3K supplémentaires de code seront pris en compte par l'éditeur de liens.

Les fonctions non transcendantes travaillent avec une précision totale de dix huit chiffres pour toutes les opérations. Les opérations transcendantes fonctionnent avec une précision d'au moins dix sept chiffres et leur précision est en général de $\pm 1 \times 10^{-18}$.

Comme les calculs BCD ne se font pas en notation infixée avec Turbo Pascal 4.0 et comme les fonctions Pascal ne peuvent pas renvoyer les types structurés (tels que des tableaux), le processus de calculs avec des réels BCD est maintenant un peu plus difficile. Toutes les routines TPBCD qui calculent des résultats réels BCD les renvoient comme des paramètres VAR. Voir la description qui suit pour les détails de syntaxe de l'appel.

Les erreurs d'exécution avec TPBCD sont gérées d'une manière directe. Toutes les fois qu'une erreur numérique se produit, division par zéro, dépassement de capacité ou opération illégale telle que racine carrée d'un nombre négatif, TPBCD génère une interruption de division par zéro (INT 0). Par défaut, le système de gestions d'erreurs de Turbo Pascal traite cette interruption, rapporte une division par zéro et arrête le programme. L'adresse de l'erreur renvoyée par Turbo Pascal sera quelque part dans l'unité TPBCD et rendra difficile la détermination de la véritable source du problème. De façon facultative, vous pouvez utiliser l'unité TPEREXEC dans les cas où les erreurs numériques de TPBCD sont récupérables. En testant *ExitCode* après la récupération d'une erreur, vous pouvez déterminer de façon plus précise l'emplacement réel de l'erreur.

TPBCD : Réels en haute précision

Cette section décrit tous les types, variables, constantes, procédures et fonctions qui se trouvent dans la partie interface de l'unité TPBCD.

Constantes

```
TailleBCD = 10;
```

Spécifie le nombre d'octets utilisés pour enregistrer un réel BCD. La valeur de 10 est homogène avec les réels BCD utilisés par Turbo Pascal 3.0. En modifiant cette constante et la constante équivalente dans TPBCDLOW.ASM, les réels BCD de plus haute précision peuvent être obtenus pour toutes les opérations sauf les opérations transcendantes (remarquez que vous aurez besoin du macro-assembleur afin de réassembler TPBCDLOW.ASM). Des réels de dix octets BCD permettent une précision de dix huit chiffres et deux chiffres supplémentaires de précision peuvent être obtenus pour chaque octet supplémentaire utilisé. Les opérations transcendantes ne gagneront pas nécessairement en précision puisque les algorithmes des fonctions sont conçus pour fournir seulement une précision de dix huit chiffres. Les algorithmes plus précis seront plus longs à exécuter et plus encombrants.

```
VirgulePourPoint : Boolean = False;
```

Si cette constante vaut "True", la fonction *Formate* renverra un '.' à la place d'une ',' et réciproquement. Cela permet d'écrire des programmes compatibles aux Etats-Unis et dans les pays qui utilisent des virgules à la place des points décimaux.

```
SigneMonetaire : Char = '$';
```

Ce caractère est utilisé pour afficher des montants avec un format dollars-et-centimes. Vous pouvez modifier ce format si vous le souhaitez, et donnez la valeur #156, par exemple, à cette constante ce qui permettra d'afficher des montants avec des livres britanniques plutôt qu'avec des dollars.

```
PiBcd : BCD = ($40, $24, $93, $58, $ 53, $26, $59, $41, $31);
```

Spécifie *Pi* dans un format BCD avec 18 chiffres.

```
ZeroBcd : DBC = ($0, $0, $0, $0, $0, $0, $0, $0, $0, $0);
```

Spécifie 0 dans le format BCD.

Types

```
BCD = array[1..TailleBCD] of Byte;
```

Format interne d'un réel BCD. Voir TPBCD.ASM pour plus de détails sur la description du format BCD.

AbsBcd

Déclaration

```
procedure AbsBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie la valeur absolue de *B1* dans *B2*.

Commentaires

Opération très rapide qui nécessite seulement d'effacer un bit du réel d'entrée et de transférer dix octets en résultat.

Exemple

```
{supposons que B1 contienne -1.2345}
AbsBcd(B1, B2);
```

B2 renvoie 1.2345

AddBcd

Déclaration

```
procedure AddBcd(B1, B2 : BCD; var B3 : BCD);
```

Objet

Additionne *B1* à *B2* et place le résultat dans B3.

Commentaires

AddBcd crée un résultat temporaire avec une précision de 19 chiffres et renvoie un résultat arrondi à 18 chiffres.

Si le résultat dépasse 1.0E64, une erreur de dépassement de capacité se produit.

Exemple

```
{supposons que B1 contienne 0.999999999999999999}
AddBCD(B1, B1, B2);
```

B2 contient alors 2 suivi de 18 zéros puisque le résultat intermédiaire contient 19 chiffres qui sont arrondis.

ArcTanBcd

Déclaration

```
procedure ArcTanBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie arc tangente de *B1* dans *B2*.

Commentaires

Comme l'opération équivalente avec des réels normaux, *ArcTanBcd* renvoie l'angle (en radians) dont la tangente est *B1*. *ArcTanBcd* renvoie toujours une valeur entre -Pi/2 et +Pi/2. Toute valeur dans l'intervalle des réels BCD est acceptable comme paramètre d'entrée.

Exemple

```
{supposons que B1 contienne 1.0}
ArcTanBCD(B1, B2);
{supposons que B3 contienne 4.0}
MultBCD(B2, B3, B4);
```

B4 retourne 3.14159265358979323, qui est la valeur de *PI*.

BcdEnReel

Déclaration

```
function BcdEnReel(B : BCD) : Real;
```

Objet

Convertit un réel BCD en réel.

Commentaires

Cette routine convertit un réel BCD en un réel de six octets, différent des réels 8087. Nous supposons que vous n'utiliserez pas l'unité BCD si vous êtes sûr de la disponibilité du composant 80x87.

Les routines convertissent un réel BCD en une chaîne ASCII de précision totale (opération rapide) et appelle la procédure *Val* de l'unité SYSTEM pour convertir la chaîne en réel. La précision de 18 chiffres d'un réel BCD est arrondie à la précision des 11 chiffres de *Turbo Pascal*.

Voir aussi

LongEntierEnBcd (TPBCD) ReelEnBcd (TPBCD)

ChnBcd

Déclaration

```
function ChnBcd(B : BCD; Largeur, Positions : Byte) : string;
```

Objet

Renvoie un réel BCD sous forme de chaîne de caractères.

Commentaires

Cette routine est fonctionnellement équivalente à la fonction de Turbo Pascal : *Str(R:Largeur:Positions,S)*. *ChnBcd* renvoie une chaîne de caractères dans un format virgule flottante (sans exposant). Remarquez que *ChnBcd* est une fonction et non une procédure comme *Str*.

Si vous voulez obtenir un réel BCD en format exponentiel, utilisez la fonction *ChnExpBcd*. La fonction *Formate* fournit également un autre moyen de formater des réels BCD. Le nombre maximum de positions décimales à spécifier est de 18, bien que *ChnBcd* en renverra plus (remplies avec des zéros) si vous le demandez. Si la largeur réelle de la chaîne dépasse *Largeur*, *ChnBcd* utilisera la largeur réelle. Si la largeur réelle est plus petite, le résultat est justifié à droite dans une chaîne de la largeur demandée.

Exemples

```
Writeln(ChnBcd(B1, 10, 2));
```

Ecrit le réel BCD *B1* de la même manière que

```
Writeln(R1:10:2);
```

le ferait pour un réel normal.

```
ChnBcd(B1 , 0, 0);
```

Retourne le réel *B1* arrondi à l'entier le plus proche dans une chaîne de largeur minimum.

Voir aussi

Formate (TPBCD)

ChnExpBcd (TPBCD)

ChnExpBcd

Déclaration

```
function ChnExpBcd(B : BCD; Largeur : Byte) : string;
```

Objet

Renvoie *B* sous forme d'une chaîne de caractères en format exponentiel.

Commentaires

Cette routine formate un réel BCD sous forme d'une chaîne en format exponentiel. Si *Largeur* est égale à 0, le résultat est une chaîne de 24 caractères de la forme - n.nnnnnnnnnnnnnnnnE+ee ; le premier caractère est un blanc pour les nombres positifs ou nuls. Si *Largeur* est différente de 0, la chaîne renvoyée aura une longueur de *Largeur* caractères. Si *Largeur* est inférieure à 24, les chiffres supplémentaires seront arrondis, si elle est plus grande que 24, la valeur sera cadrée à droite dans la chaîne et sera complétée avec des blancs en tête. Si *Largeur* est plus grande que 0 et plus petite que 8, une largeur de 8 sera utilisée.

Exemples

```
{supposons que B1 contienne 123456.9}
ChnExpBCD(B1, 0);
```

Retourne '1.23456900000000000E+05'

```
ChnExpBCD(B1, 9);
```

Retourne ' 1.23E+05'

Voir aussi

Formate (TPBCD)

ChnBcd (TPBCD)

CosBcd

Déclaration

```
procedure CosBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie le cosinus de *B1* dans *B2*.

Commentaires

Comme la fonction *Cos* de Turbo, *CosBcd* prend un angle exprimé en radians et renvoie le cosinus dans le paramètre *B2*. La plus grande valeur possible est 3.14E9 radians et la plus petite -3.14E9. La valeur du résultat est toujours comprise dans l'intervalle -1.0 à +1.0.

CosBcd est précise à $\pm 3x10^{-18}$ lorsqu'il existe une référence précise garantie. Elle correspond aux résultat fournis par un coprocesseur mathématique 80287 à $\pm 3x10^{-18}$ à partir d'un grand jeu de données.

Exemple

```
CosBcd(PiBcd, B1);
```

Retourne la valeur -0.999999999999999997 dans *B1* .

Voir aussi

SinBcd (TPBCD)

DifferentBcd

Déclaration

```
function DifferentBcd(B1, B2 : Bcd) : Boolean;
```

Objet

Renvoie "True" si *B1* <> de *B2*.

Commentaires

DifferentBcd renvoie "True" sauf si les données sont identiques sur les 18 chiffres. Utilisez ces comparaisons avec précautions.

La recherche d'une égalité approximative pourrait être définie avec une fonction telle que :

```
function NonApproxEgal(B1, B2, Epsilon : BCD) : Boolean;
{Retourne True si Abs(B1 - B2) > Epsilon}
var
   Difference : BCD;
begin
   SoustraitBCD(B1, B2, Difference);
   AbsBCD(Difference, Difference);
   NonApproxEgal := SuperieurBCD(Difference, Epsilon);
end;
```

Voir aussi

EgalBcd (TPBCD)

DivBcd

Déclaration

```
procedure DivBcd(B1, B2 : BCD; var B3 : BCD);
```

Objet

Divise *B1* par *B2* et place le résultat dans *B3*.

Commentaires

L'algorithme de division utilisé est semblable à la technique "soustraction et décalage".

Si le résultat dépasse 1.0E64, une erreur de dépassement de capacité se produit. Si le résultat est plus petit que 1.0E-63, la valeur 0 est retournée.

Voir aussi

MultBcd (TPBCD)

EgalBcd

Déclaration

```
function EgalBcd(B1, B2 : BCD) : Boolean;
```

Objet

Renvoie "True" si B1 = B2.

Commentaires

EgalBcd compare l'égalité stricte des deux opérandes. La plus petite différence entraîne un résultat égal à "False".

Pour une comparaison approximative d'égalité, vous pourriez définir une fonction telle que :

```
function ApproxEgal(B1, B2, Epsilon : BCD) : Boolean;
{Retourne true i Abs(B1 - B2) < Epsilon}
var
   Difference : BCD;
begin
   SoustraitBCD(B1, B2, Difference);
   AbsBCD(Difference, Difference);
   ApproxEgal := InferieurBCD(Difference, Epsilon);
end;
```

Pour comparer deux éléments avec une précision de un centième, vous pourriez écrire :

```
ReelEnBCD(0.001, UnDixiemeDeCentime);
if ApproxEgal(B1, B2, UnDixiemeDeCentime) then {...};
```

Voir aussi

DifferentBcd (TPBCD)

ExpBcd

Déclaration

```
procedure ExpBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie l'exponentielle de *B1* dans *B2*.

Commentaires

La plus grande valeur possible pour *B1* est 147.36. Les valeurs supérieures entraîneront une erreur de dépassement de capacité. La plus petite valeur de *B1* fournissant un résultat significatif est -145.06. *ExpBcd* renvoie 0.0 pour des données inférieures à cette valeur.

ExpBcd est précise à $\pm 1 \times 10^{-18}$ lorsqu'il existe une référence précise garantie. Elle correspond aux résultat fournis par un coprocesseur mathématique 80287 à $\pm 3 \times 10^{-18}$ à partir d'un grand jeu de données.

Exemple

```
{supposons que B1 contienne 1.0}
ExpBCD(B1, B2);
```

Retourne 2.71828182845904524 (c'est-à-dire la valeur de e) dans *B2*.

Voir aussi

LnBcd (TPBCD)

Formate

Déclaration

```
function Formate(Masque : string; B : BCD) : string;
```

Objet

Renvoie la donnée *B* sous forme d'une chaîne formatée à partir de *Masque*.

Commentaires

Formate est la caractéristique la plus importante des fonctions de formatage de Turbo 3.0 BCD. Cette fonction offre des possibilités de formatage évoluées pour les nombres réels. C'est la seule fonction de TPBCD écrite en *Pascal* ; si vous le souhaitez, vous pouvez facilement la modifier. Voir la description de la fonction *Formate* dans l'unité TPCHAINE pour plus de détails.

Voir aussi

Formate (TPCHAINE)
ChnBcd (TPBCD)
ChnExpBcd (TPBCD)

FracBcd

Déclaration

```
procedure FracBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie la partie fractionnaire de *B1* dans *B2*.

Commentaires

Le résultat de *FracBcd* est équivalent à *FracBcd* := B1 - IntBcd(B1).

Exemples

```
{supposons que B1 contienne 1.46}
FracBcd(B1, B2);
```

Retourne 0.46 dans *B2*.

```
{supposons que B1 contienne -1.46}
FracBcd(B1, B2);
```

Retourne -0.46 dans *B2*.

InferieurBcd

Déclaration

```
function InferieurBcd(B1, B2 : BCD) : Boolean;
```

Objet

Renvoie "True" si *B1* < *B2*.

Commentaires

Toutes les fonctions de comparaison BCD utilisent une routine commune de bas niveau qui effectue réellement les comparaisons et renvoie les indicateurs du 8086 en résultat. La fonction de haut niveau examine alors les indicateurs et renvoie une valeur booléenne appropriée.

La comparaison s'effectue sur la totalité des dix huit chiffres.

InferieurEgalBcd

Déclaration

```
function InferieurEgalBcd(B1, B2 : BCD) : Boolean;
```

Objet

Renvoie "True" si *B1* <= *B2*.

Commentaires

Toutes les fonctions de comparaison BCD utilisent une routine commune de bas niveau qui effectue réellement les comparaisons et renvoie les indicateurs 8086 en résultat. La fonction de haut niveau examine alors les indicateurs et renvoie une valeur booléenne appropriée.

La comparaison s'effectue sur la totalité des dix huit chiffres.

IntBcd

Déclaration

```
procedure IntBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie la partie entière de *B1* dans *B2*.

Commentaires

Si *B1* est plus grand ou égal à 0, *IntBcd* renvoie le plus grand entier plus petit ou égal à *B1*. Si *B1* est inférieur à 0, alors *IntBcd* renvoie le plus petit entier plus grand ou égal à *B1*.

Remarquez que le résultat est un réel BCD sans partie fractionnaire mais que ce n'est pas réellement un entier.

Exemple

```
{supposons que B1 contienne 1.46}
IntBcd(B1, B2);
```

Retourne 1.0 dans *B2*.

```
{supposons que B1 contienne -1.46}
IntBcd(B1, B2);
```

Retourne -1.0 dans *B2*.

LnBcd

Déclaration

```
procedure LnBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie le logarithme naturel de *B1* dans *B2*.

Commentaires

L'entrée de *LnBcd* doit être un nombre positif sinon une erreur numérique se produira. *LnBcd* est précise à $\pm 1 \times 10^{-18}$ lorsqu'il existe une référence précise garantie. Elle correspond aux résultat fournis par un coprocesseur mathématique 80287 à $\pm 2 \times 10^{-18}$ à partir d'un grand jeu de données.

Exemple

```
{supposons que B1 contienne 2.0}
LnBCD(B1, B2);
```

Retourne 6.931471800559945308E-01

Voir aussi

ExpBcd (TPBCD)

LongEntEnBcd

Déclaration

```
procedure LongEntEnBcd(L : LongInt; var B : BCD);
```

Objet

Convertit un entier long en BCD.

Commentaires

Cette routine fournit une méthode pratique pour convertir des constantes entières en valeurs BCD. Une donnée BCD peut effectivement contenir les 10 caractères du plus grand entier long.

Exemples

```
LongEntEnBcd(0, B1);
```

Place la valeur 0 dans *B1*.

```
LongEntEnBcd(MaxLongInt, B1);
```

Place 2147483647 en *B1*.

Voir aussi

ReelEnBcd (TPBCD)

MultBcd

Déclaration

```
procedure MultBcd(B1, B2 : BCD; var B3 : BCD);
```

Objet

Multiplie *B1* par *B2* et place le résultat dans *B3*.

Commentaires

L'algorithme de multiplication utilisé est similaire à la technique "multiplication avec retenue".

Si le résultat dépasse 1.0E64, une erreur de dépassement de capacité se produit. Si le résultat est plus petit que 1.0E-63, la valeur 0 est retournée.

Voir aussi

DivBcd (TPBCD)

ReelEnBcd

Déclaration

```
procedure ReelEnBcd(R : Real; var B : BCD);
```

Objet

Convertit un réel en une valeur BCD.

Commentaires

Cette procédure fournit une méthode pratique pour initialiser des constantes réelles BCD. Tant que la précision de la constante ne dépasse pas celle d'un réel de six octets, le réel BCD contiendra une valeur de constante exacte.

Si vous utilisez des constantes qui demandent une précision de plus de 11 chiffres, nous vous conseillons d'utiliser le programme de démonstration CALC fourni avec SMACS pour générer des constantes typées dans un format de tableau BCD. De cette manière, les constantes peuvent avoir une précision de 18 chiffres. D'une autre façon, vous pouvez utiliser la procédure *ValBcd* pour convertir une chaîne en une constante réelle BCD.

ReelEnBcd permet *seulement* de convertir des réels Turbo sur six octets. Elle ne permet pas de travailler avec des réels en format 8087. Nous supposons que vous n'utilisez pas l'unité TPBCD si vous pouvez êtes sûr de disposer du composant 80x87.

Exemples

```
ReelEnBcd(1.5, B1);
```

Initialise *B1* à 1.5

```
ReelEnBcd(-1.78234976E-22, B1);
```

Initialise *B1* à -1.78234976E-22

Voir aussi

LongEntEnBcd (TPBCD)

RoundBcd

Déclaration

```
function RoundBcd(B1 : BCD): LongInt;
```

Objet

Renvoie la valeur de *B1* arrondie à l'entier long le plus proche.

Commentaires

Si *B1* dépasse *MaxLongInt* (2147483647), alors *RoundBcd* déclenche une erreur de dépassement de capacité, sinon elle exécute un arrondi normal pour générer un résultat de type *LongInt*.

Si *B1* est supérieur ou égal à 0, *RoundBcd* renvoie *TruncBcd(B1+0.5)*. Sinon, elle renvoie *TruncBcd(B1-0.5)*.

Exemples

```
{supposons que B1 contienne 1.46}
RoundBcd(B1);
```

Retourne 1

```
{supposons que B1 contienne 1.50}
RoundBcd(B1);
```

Retourne 2

```
{supposons que B1 contienne -1.50}
RoundBcd(B1);
```

Retourne -2

Voir aussi

TruncBcd (TPBCD)

SinBcd

Déclaration

```
procedure SinBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie le sinus de *B1* dans *B2*.

Commentaires

Comme la fonction *Sin* de Turbo, *SinBcd* prend un angle exprimé en radians et renvoie le sinus dans le paramètre *B2*. La plus grande valeur acceptable en entrée est 3.14E9 radians et la plus petite est -3.14E9.

Exemple

```
ReelEnBCD(4.0, B1);
DivBCD(PiBCD, B1, B2); {Calcule Pi/4}
SinBCD(B2, B4);
```

Retourne Sin(PI / 4) et *B4* = 7.07106781186547525E-01

Voir aussi

CosBcd (TPBCD)

SoustraitBcd

Déclaration

```
procedure SoustraitBcd(B1, B2 : BCD; var B3 : BCD);
```

Objet

Soustrait *B2* de *B1* et place le résultat dans *B3*.

Commentaires

SoustraitBcd crée un résultat temporaire d'une précision de 19 chiffres puis l'arrondit sur 18 chiffres.

Si le résultat dépasse 1.0E64, une erreur de dépassement de capacité se produit. Si le résultat est plus petit que 1.0E-63, la valeur 0 est retournée.

SqrBcd

Déclaration

```
procedure SqrBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie le carré de *B1* dans *B2*.

Commentaires

Cette opération est équivalente à *MultBcd(B1,B1,B2)*. Elle est beaucoup plus rapide et plus facile à appeler.

Si le résultat dépasse 1.0E64, une erreur de dépassement de capacité se produit, si le résultat est plus petit que 1.0E-63, la valeur 0 est retournée.

Voir aussi

MultBcd (TPBCD)

SqrtBcd

Déclaration

```
procedure SqrtBcd(B1 : BCD; var B2 : BCD);
```

Objet

Renvoie la racine carrée de *B1* dans *B2*.

Commentaires

Les données d'entrée doivent être positives ou nulles. Sinon, une erreur numérique se produit.

L'algorithme utilisé est la méthode d'approximation de Heron. Elle est exacte sur les 18 chiffres et demande 4 itérations.

Exemple

```
{supposons que B1 contienne 2.0}
SqrtBcd(B1, B2);
```

Retourne 1.41421356237309505

SuperieurBcd

Déclaration

```
function SuperieurBcd(B1, B2 : BCD) : Boolean;
```

Objet

Renvoie "True" si *B1* > *B2*.

Commentaires

Toutes les fonctions de comparaison BCD utilisent une routine commune de bas niveau qui effectue la comparaison réelle et renvoie les indicateurs du 8086 en résultat. La fonction de haut niveau examine alors les indicateurs pour renvoyer la valeur booléenne appropriée.

SuperieurEgalBcd

Déclaration

```
function SuperieurEgalBcd(B1, B2 : BCD) : Boolean;
```

Objet

Renvoie "True" si B1 >= B2.

Commentaires

Toutes les fonctions de comparaison BCD utilisent une routine commune de bas niveau qui effectue la comparaison réelle et renvoie les indicateurs du 8086 en résultat. La routine de haut niveau examine alors les indicateurs et renvoie la valeur booléenne appropriée.

La comparaison s'effectue sur la totalité des dix huit chiffres.

TruncBcd

Déclaration

```
function TruncBcd(B1 : BCD) : LongInt;
```

Objet

Renvoie le plus grand entier long inférieur ou égal à B1.

Commentaires

Si *B1* est supérieur ou égal à 0, *TruncBcd* renvoie le plus grand entier inférieur ou égal à *B1*. Sinon *TruncBcd* renvoie le plus petit entier supérieur ou égal à B1.

Si *B1* dépasse *MaxLongInt* (2147483647), une erreur de dépassement de capacité se produit.

Exemples

```
{supposons que B1 contienne 1.46}
TruncBcd(B1);
```

Retourne 1.

```
{supposons que B1 contienne 1.999999}
TruncBcd(B1);
```

Retourne 1.

```
{supposons que B1 contienne -1.999999}
TruncBcd(B1);
```

Retourne -1.

Voir aussi

RoundBcd (TPBCD)

ValBcd

Déclaration

```
procedure ValBcd(S : string; var B : BCD; var Code : Word);
```

Objet

Convertit une chaîne de caractères en un réel BCD.

Commentaires

Cette routine fournit une routine équivalente à la routine *Val* de Turbo pour le traitement des réels BCD. *S* est une chaîne dans un format exponentiel ou en virgule flottante valide. Les blancs de tête sont acceptés mais les blancs de queue ne le sont pas.

Si la conversion se termine bien, *Code* contiendra 0 et *B* contiendra le réel en résultat. Sinon, *Code* contiendra une valeur différente de 0 qui pointe sur la position dans la chaîne d'entrée qui a entraînée l'erreur et B contiendra une valeur indéterminée. Si la chaîne mentionne un réel trop grand pour le format BCD, *Code* pointera au delà de la fin de la chaîne.

Exemples

```
ValBCD('  1.0E22  ', B1, Code);
```

Renvoie Code = 9 car les espaces de fin ne sont pas autorisés.

```
ValBCD('-1.0E99  ', B1, Code);
```

Renvoie Code = 8 car la valeur spécifiée dépasse la taille des réels BCD

```
ValBCD('0.5772156649015328606 1', B1, Code);
```

Renvoie Code = 0 et la constante de Euler dans B1. Notez que les chiffres en trop sont tronqués pour faire tenir la valeur dans la mantisse de 18 chiffres.

Voir aussi

Formate (TPBCD)
ChnBcd (TPBCD)
ChnExpBcd (TPBCD)

12 Divers

Généralités

Ce chapitre concerne les petites unités qui n'appartiennent pas réellement aux groupes généraux déjà définis. Il y a en fait deux unités de ce type (une unité et un fichier inclus). L'unité TPEREXEC permet de traiter la plupart des erreurs d'exécution. L'unité TPINLINE contient une série de "procédures inline" (nous les appellerons *macros inline*). A coté de leur propre intérêt, ces routines courtes utilisent des techniques qui pourront vous aider à écrire vos macros inline.

TPEREXEC : Gestion des erreurs d'exécution

Turbo Pascal fournit des facilités pour gérer les erreurs d'exécution. Le programmeur peut désactiver le contrôle de bornes, le contrôle de dépassement de pile et le contrôle des opérations d'entrée/sortie. Les erreurs d'allocation sur le tas peuvent être déroutées en utilisant la nouvelle possibilité de *HeapError*. Toutes les erreurs d'exécution peuvent être gérées avec des procédures écrites par l'utilisateur.

Cependant, une question reste sans réponse. Comment le programme peut-il gérer des erreurs d'exécution et se poursuivre ? *Turbo Professionnel* fournit la réponse grâce à l'unité TPEREXEC.

TPEREXEC profite des informations fournies par le gestionnaire d'erreurs générales d'exécution de *Turbo Pascal*. Le contrôle est dérouté de l'unité système normale dans une routine définie par TPEREXEC. La routine TPEREXEC distingue deux types d'erreurs : celles qui peuvent être traitées et celles qui sont fatales ; elle détermine le point de reprise du traitement après gestion de l'erreur, appelle facultativement une fonction utilisateur et rend le contrôle au programme ou au gestionnaire d'erreurs du système.

Certaines erreurs d'exécution sont irrécupérables -- TPEREXEC les reconnaît et rend le contrôle au gestionnaire d'erreurs du système qui arrête le programme proprement après avoir appelé les procédures de sortie définies par l'utilisateur. Nous considérerons maintenant qu'il y a trois types d'erreurs d'exécution fatales avec Turbo : erreur 202 (débordement de pile), erreur 203 (débordement du tas), erreurs 208 et 209 (erreurs d'overlay) et toutes les erreurs provoquées par l'utilisation du co-processeur mathématique 80x87 (de l'émulateur numérique de *Turbo Pascal* 5.0). Toutes les autres erreurs d'exécution de *Turbo Pascal*, y compris les erreurs d'entrée/sortie, peuvent être traitées.

Les problèmes de débordement de piles peuvent être résolus en augmentant la taille du segment de pile via la directive de compilation {$M} ou en réorganisant les variables locales du programme. Les erreurs de débordement du tas peuvent être traitées avec la fonction *HeapError* de *Turbo Pascal* 4.0.

Vous pouvez gérer les erreurs du co-processeur mathématique en programmant les codes d'erreur renvoyés par les statuts des registres 80x87 au lieu de laisser le co-processeur générer des conditions d'interruption. Voir l'unité TP8087 pour plus de détails.

Remarquez que les erreurs d'entrée/sortie (erreurs lors de l'exécution dont le numéro est inférieur à 200) donneront une valeur différente de 0 à *IoResult*. *IoResult* doit être initialisée (en la lisant) pour que les opérations

d'entrée/sortie puissent s'exécuter correctement. Puisque vous pouvez désactiver le contrôle des entrées/sorties de Turbo avec la directive de compilation {$I-}, il est peu vraisemblable que vous ayez à utiliser TPEREXEC pour gérer les erreurs d'entrée/sortie. Dans certaines situations, quoiqu'il en soit, le choix d'un gestionnaire d'erreurs centralisé peut être souhaitable.

Le programme qui suit met en application l'unité TPEREXEC.

```
{$R+}
program TestTpErexec;
uses
   Dos,
   TpErExec,
   TpChaine;
var
   I, J, K : integer;
   V : 1..10;

   {$F+} {Utilisation du modèle FAR}
   function RepErreur : Boolean;
   {- appelée quand une erreur survient.
      Renvoie True pour permettre la reprise -}
   var
      I : Word;
   begin
      I := IoResult;
      Assign(OutPut,'');
      rewrite(OutPut);
      writeln('Le gestionnaire d''erreurs a trouvé l''erreur  ',
      ExitCode:3, ' à l''adresse ', MotHex(Seg(ErrorAddr^)), ':' ,
         MotHex(Ofs(ErrorAddr^)));
      RepErreur := True;
   end;
   {$F-}

begin
   {Enable error recovery, and install user routine}
   RepriseErreur(True,@RepErreur);

   I := 0; J := 1; K := J div I;
   if ExitCode <> 0 then writeln('Erreur : division par zéro');

   I := 20; V := I;
   if ExitCode <> 0 then writeln('Erreur d''intervalle');
end.
```

Variables

```
ReprisePossible : Boolean;
```

Puisque TPEREXEC doit modifier le programme Turbo, son code d'initialisation recherche soigneusement les instructions à modifier. Si tout va bien, TPEREXEC initialise *ReprisePossible* à "True". Actuellement, il y a seulement deux raisons pour que *ReprisePossible* ait la valeur "False" : soit la bibliothèque SYSTEME de Turbo gère très différemment les erreurs, soit le programme de l'application a pris le contrôle de INT $0 (division par 0) ou INT $23 (Ctrl break) avant que les initialisations de TPEREXEC aient été exécutées. Le premier problème nécessite de modifier l'initialisation de TPEREXEC. Le deuxième peut être résolu en redisposant les instructions USES et en plaçant TPEREXEC avant toutes les unités définies par l'utilisateur.

RepriseErreur

Déclaration

```
procedure RepriseErreur(On : Boolean; RoutineUtl : Pointer);
```

Objet

Active ou désactive la gestion des erreurs et appelle une routine utilisateur de traitement d'erreurs.

Commentaires

Appelez cette procédure pour activer ou désactiver la gestion des erreurs d'exécution. Si le paramètre *On* vaut "True", la gestion des d'erreurs est activée, sinon elle est désactivée et le système normal de gestion des erreurs normales de *Turbo Pascal* est utilisé.

TPEREXEC permet de gérer la plupart des erreurs d'exécution mais pas toutes. Voir l'introduction de TPEREXEC pour plus de détails.

RoutineUtl est un pointeur sur une fonction qui permet de personnaliser la gestion des erreurs dans une application. Si vous ne désirez pas définir une telle fonction, passez simplement la valeur *Nil* dans ce paramètre.

RoutineUtl doit pointer sur une fonction déclarée comme suit :

```
{$F+} {Utilisation du modèle FAR}
function RepErreur : Boolean;
{- appelée quand une erreur survient.
   Renvoie True pour permettre la reprise -}
begin
   ...
end;
{$F-}
```

Remarquez que la routine doit être compilée avec l'option FAR. Si vous ne faites pas précéder la fonction par la directive {F+}, un arrêt brutal du programme se produira.

La fonction *RoutineUtl* sera appelée toutes les fois qu'une erreur d'exécution récupérable apparaîtra. A ce moment, l'octet global *ExitCode* contiendra le numéro de l'erreur d'exécution et le pointeur global *ErrorAddr* contiendra l'adresse normalisée de l'erreur. *RoutineUtl* peut évaluer ces variables et effectuer le traitement approprié. Avant de sortir, *RoutineUtl* prendra la valeur "True" si le traitement est souhaité et "False" si le programme doit être stoppé.

Si une erreur d'exécution ne peut être traitée ou si *RoutineUtl* renvoie "False", TPEREXEC passe le contrôle au gestionnaire d'erreurs système. Ce gestionnaire appellera normalement les procédures de sortie utilisateur, fermera tous les fichiers, restaurera les vecteurs d'interruption et en stoppera l'exécution.

Si une erreur d'exécution peut être traitée et que *RoutineUtl*, si elle existe renvoie "True", les routines de TPEREXEC redonneront le contrôle à l'instruction machine qui suit immédiatement celle qui a provoqué l'erreur. Le programme de l'application est responsable de l'examen de l'octet *ExitCode* qui indique si une erreur s'est produite. *ExitCode* contiendra les valeurs normales associées aux erreurs d'exécution de Turbo : 0 en cas de succès et une valeur différente de 0 en cas d'interruption. Voir le manuel *Turbo Pascal* pour la liste détaillée des codes d'erreur.

Toutes les fois qu'une application utilise le mécanisme de gestion d'erreur de *Turbo Professionnel*, TPEREXEC appelle automatiquement *RepriseErreur* pour activer le mécanisme de gestion d'erreurs. Par défaut, TPEREXEC ne génère pas *RoutineUtl*.

Exemple

```
{$F+}
function RepErreur : Boolean;
var
   I : Word;
begin
   I := IOResult;          {Efface IOResult}
   RepErreur := True;      {Reprise de toutes les erreurs}
end;
{$F-}

begin
   RepriseErreur(True, @RepErreur);
   ...
end.
```

Exemple simple de *RoutineUtl*. La routine efface *IoResult* pour permettre aux opérations d'entrée/sortie de se poursuivre et renvoie le résultat "True" pour traiter toutes les formes d'erreurs d'exécution.

Voir aussi

Erreur8087 (TP8087)

Exceptions8087 (TP8087)

TPINLINE : Macros inline

Les procédures inline appartiennent aux nouvelles caractéristiques très puissantes de *Turbo Pascal* 4.0. Contrairement aux procédures et fonctions normales qui résident à une adresse fixe en mémoire, le contenu tout entier d'une procédure inline est inséré directement dans le code objet produit par le compilateur chaque fois qu'il est référencé. Bien que les procédures inline puissent accepter des paramètres et renvoyer des résultats de fonctions comme les fonctions et procédures normales, le compilateur les traite plus comme des macros symboliques telles que celles disponibles en langage 'C'. Pour ces raisons, nous préférons les appeler *macros inline.*

Puisque les macros inline appartiennent aux nouveautés de *Turbo Pascal,* nous avons décidé d'en inclure quelques unes dans *Turbo Professionnel,* en partie pour montrer comment les écrire, mais aussi pour mettre en évidence des mécanismes très pratiques.

TPINLINE contient ces macros inline. Entre autres choses, vous trouverez des routines pour :

- Positionner une étiquette de référence vers laquelle vous pourez revenir depuis n'importe où dans votre programme
- Appeler toute procédure ou fonction sans paramètre, pour construire aisément des tables de saut
- Manipuler les mots individuels qui composent un entier long, comme avec les fonctions *Hi, Lo* et *Swap* qui permettent de traiter séparément les octets dans un mot
- Normaliser des pointeurs
- Initialiser, effacer et tester des indicateurs binaires
- Changer des variables sans utiliser des variables temporaires
- Initialiser un tableau de n'importe quel type sans utiliser une boucle FOR.

Contrairement aux autres unités qui composent la bibliothèque *Turbo Professionnel,* TPINLINE.INC possède des possibilités d'évolution sans limite. Vous devrez donc consulter le fichier ALIRE sur la disquette d'origine afin de contrôler l'ajout de nouvelles routines depuis la dernière impression de la documentation.

Types

```
EnregSaut = record
            RegSP, RegBP : Word; PtSaut : Pointer;
        end;
```

Utilisé par *InitSaut* et *SautLong* afin d'enregistrer des données utilisées pour exécuter des branchements non locaux.

AppelAncISR

Déclaration

```
procedure AppelAncISR(AncIsr : Pointer);
```

Objet

Appelle la routine ISR précédente à partir d'une procédure d'interruption.

Commentaires

Simule une interruption en empilant les indicateurs CPU et en effectuant un appel FAR à l'adresse passée dans *AncIsr*. Comme *EmuleInt* (TPINT), cette macro permet d'appeler l'ISR associée à un vecteur d'interruption donné avant que vous le capturiez. Contrairement à *EmuleInt*, elle est destinée aux ISR écrites avec des instructions inline, plutôt qu'en Pascal. Remarquez que le registre BX est détruit par cette macro.

Exemple

```
GetIntVec($09, SauveInt09);
...
procedure MonISR(BP : Word); Interrupt;
begin
   AppelAncISR(SauveInt09);
   {... Suite ...}
end;
```

Appelle le précédent gestionnaire d'interruption $09.

Voir aussi

EmuleInt (TPINT)

SautAncIsr (TPINLINE)

AppelFar

Déclaration

```
procedure AppelFar(AddrProc : Pointer);
```

Objet

AddrProc est l'adresse de la routine appelée avec l'option FAR.

Commentaires

Cette macro permet d'appeler n'importe quelle procédure dans n'importe quel segment de code, tant (1) qu'elle ne compte aucun paramètre, (2) qu'elle a été compilée avec l'option FAR (sélectionnée avec la directive de compilation {$F+}) et (3) qu'elle n'est pas imbriquée dans une autre procédure. *AppelFar* peut être utilisée par exemple pour implémenter une table de sauts. Evidemment cette macro n'est pas utile avec Turbo 5.0 qui met en oeuvre les variables de type procédure comme c'est le cas dans la version 4.0.

Exemple

```
PtrProc := @MaProcedure;
...
AppelFar(PtrProc^);
```

Provoquera un appel de *MaProc,* avec l'option FAR.

```
var TableDeSaut : array[1..10] of Pointer;
...
AppelFar(TableDeSaut[Indice]);
```

Provoquera un appel avec l'option FAR de la procedure dont l'adresse est rangée dans *MaTableSaut[Indice].*

Voir aussi

AppelNear (TPINLINE)

AppelNear

Déclaration

```
procedure AppelNear(OffsetProc : Word);
```

Objet

OffsetProc est l'offset de la routine appelée avec l'option near.

Commentaires

Cette routine permet d'appeler n'importe quelle procédure dans le segment de code courant tant (1) qu'elle ne comporte aucun paramètre, (2) qu'elle a été compilée avec l'option NEAR (sélectionnée avec la directive de compilation {$F-} option par défaut) et (3) qu'elle n'est pas imbriquée dans une autre procédure. Elle peut être utilisée, par exemple, pour implémenter une table de sauts.

Exemple

```
var
   TableSaut : array[1..10] of Word;
...
AppelNear(TableSaut[I]);
```

Provoque un appel avec l'option NEAR de la procédure dont l'offset est rangé dans *MaTableSaut[I]*.

Voir aussi

AppelFar (TPINLINE)

EchangeMots

Déclaration

```
function EchangeMots(var I, J :word) ;
```

Objet

Echange les mots *I* et *J*.

Commentaires

Cette procédure vous permet d'échanger les valeurs de I et J sans utiliser de variables temporaires. Elle est très utile dans les tris.

Exemple

```
if MonTableau[I] > MonTableau[I+1] then
   EchangeMots(MonTableau[I], MonTableau[I+1]);
```

Echange *MonTableau[I]* avec *MonTableau[I+1]* si *MonTableau[I]* est supérieur.

Voir aussi

EchangeOctets (TPINLINE) EchangeStructs (TPINLINE)

EchangeOctets

Déclaration

```
procedure EchangeOctets(var I, J : Byte);
```

Objet

Echange les octets I et J.

Commentaires

Cette macro vous permet d'échanger les valeurs de I et J sans utiliser de variables temporaires. Elle est très utile dans les tris.

Exemple

```
if MonTableau[I] > MonTableau[I+1] then
   EchangeOctets(MonTableau[I], MonTableau[I+1]);
```

Echange *MonTableau[I]* et *MonTableau[I+1]* si *MonTableau[I]* est supérieur.

Voir aussi

EchangeMots(TPINLINE)

EchangeStructs

Déclaration

```
procedure  EchangeStructs(var I, J ; Taille : Word);
```

Objet

Echange les structures I et J.

Commentaires

Cette macro vous permet d'échanger les valeurs de I et J sans utiliser de variables temporaires. I et J sont des variables de tout type, mais doivent avoir la même taille. Elle est très utile dans les tris.

Exemple

```
if MonEnreg1.Champ > MonEnreg2.Champ then
   EchangeStructs(MonEnreg1, MonEnreg2, SizeOf(MonEnreg1));
```

Echange *MonEnreg1* et *MonEnreg2* si *MonEnreg1.Champs* est supérieur à *MonEnreg2.Champs*.

```
if MaChaine1 > MaChaine2 then begin
   I := MotMax(Length(MaChaine1), Length(MaChaine2));
   EchangeStructs(MaChaine1, MaChaine2, I+1);
end;
```

Echange *MaChaine1* et *MaChaine2* si *MaChaine1* est supérieure.

Remarquez que le nombre d'octets déplacés est limité par la longueur de la plus longue des chaînes plus 1 (pour tenir compte de la longueur en octets des deux chaînes).

Voir aussi

EchangeMots(TPINLINE) MotMax(TPINLINE)

IndicateurExiste

Déclaration

```
function IndicateurExiste(Indicateur, MasqueIndicateur : Word):Boolean;
```

Objet

Renvoie "True" si le bit spécifié par *MasqueIndicateur* est à 1 dans *Indicateur*.

Commentaires

Cette macro vérifie si les bits qui se trouvent dans *MasqueIndicateur* sont également positionnés dans *Indicateur*. Dans la plupart des cas, *MasqueIndicateur* ne devra pas avoir plus d'un bit à 1. *IndicateurExiste* peut indiquer si un bit est à 1 ou si aucun ne l'est, mais pas si tous le sont.

Exemples

```
if IndicateurExiste(Mot, $03) then {};
```

Probablement incorrect. Ceci ne vous indiquera pas si les bits 0 et 1 sont à 1 tous les deux ($03 = 0000000000000011 en binaire). Vous saurez simplement si l'un des bits est positionné à 1.

```
if IndicateurExiste(Mot, $02) then {};
```

Correct. Ceci retournera True si le bit 1 est à 1 dans *Mot* ($02 = 0000000000000010 en binaire).

Voir aussi

ZeroIndicateur (TPINLINE) InitIndicateur (TPINLINE)

InitIndicateur

Déclaration

```
procedure InitIndicateur(var Indicateur : Word;
                         MasqueIndicateur : Word);
```

Objet

Positionne les bits de *Indicateur* qui existent dans *MasqueIndicateur.*

Commentaires

Cette macro met à 1 tous les bits de *Indicateur* qui existent dans *MasqueIndicateur*. Remarquez que le paramètre *Indicateur* doit être une variable de type Word.

Exemple

```
InitIndicateur(Mot, $02);
```

Met à 1 le bit 1 dans *Mot* ($02 = 0000000000000010 en binaire).

```
InitIndicateur(Mot, $07);
```

Met à 1 les bits 2, 1 et 0 dans *Mot* ($07 = 0000000000000111 en binaire).

Voir aussi

ZeroIndicateur (TPINLINE) IndicateurExiste (TPINLINE)

InitSaut

Déclaration

```
procedure InitSaut(var DestSaut : EnregSaut);
```

Objet

Sauvegarde les registres SP et BP actuels et une adresse de saut.

Commentaires

Cette macro sauvegarde les données relatives à la pile actuelle et le pointeur d'instruction donné, qui pourront être utilisés par *SautLong* pour relancer l'exécution à l'instruction suivant l'appel de *InitSaut*. Elle peut être utilisée principalement avant la boucle d'un programme principal. Vous pourrez revenir au début de la boucle en utilisant *SautLong* quelque soit le moment où l'erreur s'est produite.

Exemple

```
var
   MonEnregSaut : EnregSaut;
...
InitSaut(MonEnregSaut);
WriteLn('Voici JumpDest.PtSaut');
{... Suite ...}
SautLong(MonEnregSaut);
```

L'appel à *InitSaut* permettra d'enregistrer la donnée nécessaire à *SautLong* pour relancer l'exécution à l'adresse de l'instruction *WriteLn*.

Voir aussi

SautLong (TPINLINE)

MotMax

Déclaration

```
function MotMax(A,B : Word) : Word;
```

Objet

Renvoie le maximum de *A* et *B*.

Commentaires

Cette macro compare les deux valeurs *A* et *B* non signées, et renvoie la plus grande des deux valeurs.

Exemple

```
I := MotMax(12, 40000);
```

I aura la valeur 40000.

Voir aussi

MotMin(TPINLINE)

MotMin

Déclaration

```
function MotMin(A, B ; Word) : Word;
```

Objet

Renvoie le minimum de A et B.

Commentaires

Cette macro compare deux valeurs A et B non signées, et renvoie la plus petite des deux valeurs.

Exemple

```
I := MotMin(12, 40000);
```

I aura la valeur 12.

Voir aussi

MaxMot(TPINLINE)

MotPoidsFaible

Déclaration

```
function MotPoidsFaible(L : LongInt) : Word;
```

Objet

Renvoie le mot de poids faible de *L*.

Commentaires

Cette macro renvoie le mot de poids faible d'un entier long. La routine équivalente existe dans la version *Turbo Pascal* Macintosh mais pas dans la version IBM.

Exemple

```
const
   EntierLong = $FFFF0000;
...
Mot := MotPoidsFaible(EntierLong);
```

Affecte $0000 à *Mot*.

Voir aussi

MotPoidsFort (TPINLINE) PermuteMot (TPINLINE)

MotPoidsFort

Déclaration

```
function MotPoidsFort(L : LongInt) : Word;
```

Objet

Renvoie le mot de poids fort de *L*.

Commentaires

Cette fonction renvoie le mot de poids fort d'un entier long. La routine équivalente existe dans la version *Turbo Pascal* Macintosh mais pas dans la version IBM.

Exemple

```
const
   EntierLong = $FFFF0000;
...
Mot := MotPoidsFort(EntierLong);
```

Affecte $FFFF à *EntierLong*.

Voir aussi

MotPoidsFaible (TPINLINE) EchangeMots (TPINLINE)

Normalise

Déclaration

```
function Normalise(P : Pointer) : Pointer;
```

Objet

Renvoie *P* comme un pointeur normalisé.

Commentaires

Cette macro normalise un pointeur. C'est-à-dire qu'elle calcule le segment et l'offset de telle façon que l'offset appartienne à l'intervalle $0000:$000F.

Exemple

```
P1 := Ptr($0000, $0410);
P2 := Normalise(P1);
```

Fait pointer *P2* sur $0410:$0000.

```
P1 := Ptr($0040,$0113);
P2 := Normalise(P1);
```

Fait pointer *P2* sur $0051:$0003.

PermuteMot

Déclaration

```
procedure PermuteMot(L : LongInt) : LongInt;
```

Objet

Echange les mots de poids fort et de poids faible de *L*.

Commentaires

Cette macro échange les mots de poids faible et de poids fort d'un entier long. Elle existe dans la version Macintosh de *Turbo Pascal,* mais pas dans la version IBM.

Exemple

```
const
   EntierLong = $FFFF0000;
...
L := PermuteMot(EntierLong);
```

Affecte $0000FFFF à *L*.

Voir aussi

MotPoidsFort (TPINLINE) MotPoidsFaible (TPINLINE)

RemplitMot

Déclaration

```
procedure RemplitMot(var Dest ; Compte, Filler : Word);
```

Objet

Remplit la mémoire à partir de l'adresse *Dest* avec *Compte* occurrences de *Filler*.

Commentaires

Cette macro est semblable à la procédure standard *FillChar* mais elle remplit la mémoire avec une valeur exprimée en mots et non en octets. *Dest* est la zone mémoire à remplir, *Filler* est la valeur du mot avec lequel la zone doit être remplie, et *Compte* est le nombre de mots à remplir. *RemplitMot* est très utile pour initialiser des tableaux. Remarquez qu'aucun contrôle de bornes n'est effectué pour éviter des débordements de segments.

Exemple

```
var
   MonTableau1 : array[1..100] of Word;
   MonTableau2 : array[1..100] of Integer;
   I : Integer;
...
RemplitMot(MonTableau1, 100, 1);
I := 1;
RemplitMot(MonTableau2, 100, Word(I));
```

Initialise les 100 éléments de MonTableau1 et MonTableau2 avec 1.

RemplitStruct

Déclaration

```
procedure RemplitStruct(var Dest ; Compte : Word ; var Filler;
                        TailleFiller : Word);
```

Objet

Remplit la mémoire à partir de l'adresse *Dest* avec *Compte* occurrences de *Filler*.

Commentaires

Cette macro est semblable à la procédure standard *FillChar*, mais remplit la mémoire avec une valeur de *TailleFiller* caractères plutôt qu'avec une valeur de 1 caractère. *Dest* est l'adresse de la zone mémoire à remplir. *Filler* est la valeur de la donnée qui sert à remplir la zone. Sa taille est donnée par *TailleFiller*. Compte est le nombre d'occurrences de *Filler*. *RemplitStruct* est très utile pour initialiser des tableaux. Notez qu'il n'y a pas de contrôle de bornes pour éviter les débordements de segments.

Exemple

```
var
   MonTableau : array[1..1000] of Real;
const
   R : Real = 1.2345;
...
RemplitStruct(MonTableau, 1000, R, SizeOf(R));
```

Initialise les mille éléments de *MonTableau* à 1.2345.

SautAncIsr

Déclaration

```
procedure SautAncIsr(AncIsr : Pointer);
```

Objet

Saut vers l'ISR précédente à partir d'une procédure interrompue.

Commentaires

Comme *ChainInt* (TPINT), cette macro vous permet de restaurer des registres CPU empilés au moment où la procédure a été interrompue puis d'appeler l'ISR associée à un vecteur donné, avant que vous ne le capturiez. Contrairement à *ChainInt*, elle est destinée aux ISR écrites avec des instructions inline plutôt qu'en Pascal.

Exemple

```
GetIntVec($09, SauveInt09);
...
procedure MonISR(BP : Word); interrupt;
begin
   {... Suite ...}
   SautAncIsr(SauveInt09^);
end;
```

Incorrect ! Le '^' n'est pas nécessaire.

```
SautAncIsr(SauveInt09);
```

Saut vers la routine d'interruption $09 précédente.

Voir aussi

AppelAncIsr (TPINLINE) ChainInt (TPINT)

SautLong

Déclaration

```
procedure SautLong(var DestSaut : EnregSaut);
```

Objet

Restaure SP, BP et saute à *DestSaut.PtSaut*.

Commentaires

Cette macro utilise les données enregistrées dans *EnregSaut* par *InitSaut* pour effectuer un saut. Elle restaure la pile à la condition existant avant l'appel de *InitSaut*, puis saute à l'instruction suivant l'appel.

Très important : *SautLong* ne peut être utilisée pour passer d'un cadre de pile supérieur à un cadre de pile inférieur. Considérons le cas suivant par exemple :

```
program MonProg;
   Procedure MaProc;
   begin
      {Pointe B}
   end;
begin
   {Pointe A}
end.
```

Il sera parfaitement possible d'appeler *InitSaut* au point *A* et plus tard d'appeler *LongSaut* depuis le point *B* pour revenir au point *A*. Vous ne pourrez pas, quoiqu'il en soit, faire l'inverse (sauter en *B* à partir de *A*). Bien que *SautLong* utilise une instruction de saut FAR, l'option FAR {$F+} comparée à l'option NEAR {$F-} est impossible, comme avec les instructions d'appel.

Exemple

```
var
   MonEnregSaut : EnregSaut;
...
InitSaut(MonEnregSaut);
WriteLn('Voici JumpDest.PtSaut');
{... Suite ..}
SautLong(MonEnregSaut);
```

Voir aussi

InitSaut (TPINLINE)

ZeroIndicateur

Déclaration

```
procedure ZeroIndicateur(var Indicateur : Word;
                         MasqueIndicateur : Word);
```

Objet

Met à zéro les bits spécifiés par *MasqueIndicateur* dans *Indicateur*.

Commentaires

Cette macro met à zéro tous les bits du mot *Indicateur* qui correspondent aux bits *MasqueIndicateur*. Remarquez que le paramètre *Indicateur* doit être une variable de type Word.

Exemple

```
ZeroIndicateur(IndicsBit, $02);
```

Efface le bit 1 dans *IndicsBit* ($02 = 0000000000000010 en binaire)

Voir aussi

IndicateurExiste (TPINLINE)

InitIndicateur (TPINLINE)

13 Programmes de démonstration

Comme vous allez le découvrir, *Turbo Professionnel* contient un nombre impressionnant de programmes de démonstration. Sauf dans deux cas où notre enthousiasme nous a emportés (DEROULFE, par exemple), nous avons essayé non seulement de fournir des exemples réalistes, mais aussi de vous procurer des programmes réellement utiles. Nous pensons réellement que ces programmes vous aideront beaucoup plus que des petits programmes illustrant le mode d'emploi des unités. Comme le montre la couverture, il s'agit d'un logiciel professionnel et nous pensons que nos programmes de démonstration vous le prouveront.

SMACS : Processeur de macros

SMACS est un processeur de macros-clavier résident en mémoire, semblable à SuperKey de Borland. Pour charger SMACS, tapez :

SMACS [FichierMacro]

où *FichierMacro* est un paramètre optionnel désignant le nom d'un fichier contenant des macros précédemment enregistrées. SMACS est livré avec un fichier de macros exemples, TURBO5.MAC que vous pouvez utiliser avec l'éditeur Turbo.

Une fois que SMACS est chargé, vous pouvez l'appeler à tout moment en pressant sur (ALT)-(SHIFT)-S (shift gauche). Vous verrez apparaître le menu suivant :

Les commandes suivantes sont disponibles à partir du menu principal :

Edite

Modifie une macro existante ou crée une nouvelle macro. Lorsque vous exécutez cette commande, vous devez pressez la touche associée à la macro et ensuite le nom (facultatif) de la macro. Une fois dans l'éditeur de macros, vous pouvez parcourir les touches existantes avec les flèches. Par défaut, l'éditeur de macros est en mode d'insertion. Lorsque le curseur a été positionné, vous pouvez insérer de nouvelles touches en appuyant simplement sur ces touches. La touche (INS) active ou désactive le mode Insertion. (DEL) et (BACKSPACE) effacent les touches macros. Si vous devez insérer une ou plusieurs touches qui ont une fonction spéciale à l'intérieur de l'éditeur de macros, (ESC), (ENTER), (INS), (DEL), les flèches, etc -- appuyez sur (SCROLL LOCK) pour activer le mode Littéral puis appuyez sur les touches. Pour revenir au mode Commande, appuyez à nouveau sur (SCROLL LOCK). Vous pouvez terminer une session de modifications en appuyant sur (ENTER) pour sauvegarder vos modifications ou sur (ESC) pour les annuler. Pour plus de détails sur l'éditeur de macros, voir la partie concernée de l'unité TPEDMAC.

Supprime

Efface une macro précédemment définie ou efface toutes les macros. Un message vous demandera si vous voulez supprimer toutes les macros. Si vous répondez en appuyant sur O, vous devrez confirmer votre réponse. Si vous

appuyez sur N ou (ENTER), vous devrez appuyer sur la touche de la macro à supprimer. Vous pouvez appuyer sur la touche appropriée ou (CTRL)-é pour annuler l'opération.

Lit

Charge un fichier macro en mémoire. Si des macros sont déjà définies, vous devrez confirmer l'effacement de ces macros avant que les nouvelles soient lues. Remarquez que vous pouvez également charger un fichier macro à partir d'une ligne de commandes DOS ou à partir d'un fichier batch. Par exemple :

```
cd\turbo
smacs turbo5.mac
turbo
```

chargera les macros exemples de TURBO5.MAC et effacera les macros existantes *sans demande de confirmation.*

écRit

Sauvegarde les macros en cours de définition dans le fichier de macros. Si le fichier existe déjà, vous devrez confirmer si vous voulez l'écraser.

lIst

Affiche une liste de toutes les macros déjà définies. Les touches (↑) et (↓) déroulent la liste s'il y a plus de macros que l'écran ne peut en afficher en une fois. (HOME) et (END) vous placent respectivement au début et à la fin de la liste et (ESC) vous renvoie au menu principal.

sTatut

Affiche le nombre de macros qui sont déjà définies et le total de mémoire disponible pour de nouvelles macros.

Configure

Affiche un menu qui vous permet de voir et de modifier la configuration de SMACS.

Menu principal

Change la hotkey utilisée pour activer le menu principal de SMACS.

Début de macro

Change la hotkey utilisée pour activer l'enregistrement des macros.

Fin de macro

Change la hotkey utilisée pour désactiver l'enregistrement de macros.

Lecteur d'écran

Change la hotkey qui active le lecteur d'écran.

Mémoire macro

Change la mémoire disponible pour les macros.

Délai de répétition

Spécifie l'intervalle de temps utilisé pour relancer les macros. Si l'intervalle de temps est différent de 0, SMACS attendra le nombre de millisecondes spécifié avant de renvoyer une touche particulière dans la macro. De tels intervalles de temps sont quelquefois nécessaires pour éviter la perte de touches. La valeur par défaut est 0, la valeur maximum est 100.

Répétition de macros

Activation ou désactivation de l'exécution des macros.

Reconfiguration du clavier

Si cette option est positionnée, vous pouvez affecter des macros aux touches du clavier telle que (F11) (sur un clavier étendu), et (ALT)-(↑) qui normalement ne pourraient être associées avec des macros. Voir la documentation de l'unité TPCLPLUS pour une liste complète des nouvelles touches que cette option permet d'utiliser.

Sauvegarde de la configuration

Sauvegarde les paramètres actuels de façon permanente. SMACS.EXE doit être dans (1) le répertoire courant, (2) le lecteur et le répertoire à partir duquel il a été chargé, (si vous utilisez DOS 3.x ou version supérieure) ou (3) un répertoire sur le PATH du DOS.

Utilisez les touches (↑) et (↓) pour vous déplacer sur les options et appuyez sur (ENTER) pour la sélectionner. Les quatre premières options permettent de modifier une des hotkeys de SMACS. Il y a certaines limites sur les hotkeys que vous pouvez affecter de cette façon, mais vous pouvez les mettre en oeuvre avec ces commandes de n'importe quelle façon. Sauf pour le paramétrage de la mémoire Macro, toutes les modifications dans le menu de configuration seront immédiatement effectives. En supposant que vous utilisez la commande de configuration *Sauve* pour sauvegarder vos modifications, la nouvelle valeur de la mémoire réservée aux macros sera effective la prochaine fois que SMACS sera chargé.

eNlève

Permet dans de nombreux cas de décharger SMACS de la mémoire sans avoir à relancer le système. Il ne pourra pas être déchargé si un programme

a été chargé "au-dessus" de lui et qu'il utilise les mêmes vecteurs d'interruptions. Par conséquent, d'une façon générale, vous ne pourrez le décharger à moins que ce ne soit le dernier programme en mémoire. Vous pouvez aussi décharger SMACS à partir de la ligne de dommandes en entrant : **SMACS -U**

cApture

Permet de capturer l'écran tout entier au-dessous du menu SMACS et de le sauvegarder sur un fichier.

imP

Soumet un fichier à la commande d'impression simultanée de DOS 3.x ou version supérieure. Si PRINT.COM n'est pas résident ou si vous utilisez DOS 2.x, un message d'erreur sera affiché.

En plus de ces commandes disponibles sur le menu principal, SMACS possède trois autres commandes qui sont disponibles par hotkey (affichées dans la deuxième colonne).

Début de macro (ALT)-D

Active l'enregistrement de macros. Vous devrez entrer la touche affectée à la macro. Si une macro a déjà été affectée à cette touche, vous devrez préciser si la macro existante doit être écrasée. De même, si vous spécifiez une touche telle que **<A>** ou **<+>**, utilisée pour la frappe normale, vous devrez confirmer votre choix.

Fin de macros (ALT)-F

Désactive l'enregistrement de macros. Vous devrez préciser le nom à affecter à la macro. Si vous ne souhaitez pas assigner un nom à la macro, appuyez simplement sur **Enter**.

Lecteur d'écran (ALT)-( )

Active le lecteur d'écran. Le lecteur d'écran permet de capturer une partie de l'écran actuel (un bloc d'au plus 1000 caractères) et de l'enregistrer dans une macro. Utilisez les flèches pour positionner le curseur puis appuyez sur (ENTER) pour délimiter le début du bloc de caractères. Puis déplacez le curseur à la fin du bloc et appuyez sur (ENTER) une seconde fois pour le sélectionner. Lorsque vous déplacez le curseur, l'écran est mis en inversion vidéo pour indiquer les limites actuelles du bloc. Une fois que le bloc a été marqué et sélectionné, vous devrez entrer la touche à affecter à la macro. Pour annuler l'opération avant que le bloc soit sélectionné, appuyez sur (ESC).

Remarquez que seulement l'une des quatre commandes disponibles avec les hotkeys peut être utilisée à la fois. Vous ne pouvez pas, par exemple, activer le lecteur d'écran à partir du menu principal.

CALC : Calculatrice du programmeur

CALC est une calculette résidente en mémoire, spécialement conçue pour les programmeurs. Elle peut fonctionner dans cinq modes différents, deux pour les calculs en virgule flottante et trois pour les calculs sur les entiers et elle peut exécuter onze opérations arithmétiques et logiques différentes. Les calculs sur les entiers utilisent les entiers longs appartenant à l'intervalle -2147483648 ($80000000) à 2147483647 ($7FFFFFFF). Les calculs en virgule flottante sont exécutés en utilisant les réels BCD (Binary Coded Decimal) avec une précision de 18 chiffres et une étendue de 1E-63 à 1E+63.

Pour activer CALC, appuyez sur (ALT)-(SHIFT)-C (shift gauche). Ce que vous devrez voir dans le coin supérieur droit de l'écran aura l'apprence suivante :

```
= CALC : Calculatrice du programmeur 5.00
                                45.67
                              1.9E-12 *
                   0.000000000086773 -
- Flot ------------------------ Sauve -----
                                    0
- Commandes ------------------------------
 Clr effEnt Bin Dec Hex Flot exP Touche
 Sauve Insère And Mod Not Or Xor shL shR
==========================================
```

Les trois lignes supérieures représentent la fenêtre de calculs et sont utilisées pour entrer les nombres et afficher les résultats des calculs. La seconde ligne est toujours utilisée pour entrer les nombres et la troisième est toujours utilisée pour afficher les résultats. Pour exécuter un calcul, suivez ces étapes :

1. Entrez le premier nombre.
2. Appuyez sur la touche correspondante à une opération logique ou arithmétique (+, -, *, /, A, M, N, O, X, L, ou R).
3. Entrez le deuxième nombre.
4. Appuyez sur (ENTER) ou = pour afficher le résultat.

   ou

   Appuyez sur l'une des touches de l'étape 2 pour terminer le calcul et utiliser le résultat comme première valeur d'un deuxième calcul.

Une fois que vous avez obtenu un résultat, vous pouvez commencer un nouveau calcul en entrant un nombre ou appuyer sur l'une des touches de l'étape 2 pour utiliser le résultat comme première valeur d'un deuxième calcul. Remarquez qu'il n'est pas nécessaire d'appuyer sur (NUM LOCK) avant

d'entrer les nombres sur le pavé numérique. Le programme CALC active automatiquement la touche (NUM LOCK) lorsqu'il est lancé et restaure l'état précédent de la touche (NUM LOCK) à la fin.

Les deux lignes inférieures affichent un menu des commandes disponibles. Au-dessus, se trouve la valeur actuelle de la valeur du buffer sauvegardé. Il existe deux buffers, un pour les valeurs réelles et un pour les entiers.

Les commandes suivantes sont disponibles avec CALC :

Commandes de base de la calculette

C	**Clr**	Efface toute la fenêtre de calculs.
E	**effEnt**	Efface l'entrée actuelle (la seconde ligne de la fenêtre de calculs).
S	**Sauve**	Enregistre l'entrée actuelle dans le buffer approprié.
I	**Insère la valeur enregistrée**	La valeur enregistrée dans le buffer devient l'entrée actuelle.

Commandes de mode

B	**Binaire**	Active le mode binaire (entier).
D	**Décimal**	Active le mode décimal (entier).
H	**Hexadécimal**	Active le mode hexadécimal (entier).
F	**virgule Flottante**	Active le mode virgule flottante normale (réel). Les résultats sont affichés sans exposant lorsque cela est possible. Par exemple '0.0057' au lieu de '5.7E-03'.
P	**exPosant**	Active le mode virgule flottante avec exposant (réel). Les résultats sont toujours affichés avec un exposant. Par exemple, '5.7E-03' au lieu de '0.0057'.

Opérateurs arithmétiques

+	**Addition**	Ajoute la valeur 1 et la valeur 2.
-	**Soustraction**	Soustrait la valeur 2 de la valeur 1.
*	**Multiplication**	Multiplie la valeur 1 par la valeur 2.
/	**Division**	Divise la valeur 1 par la valeur 2.
M	**Modulo**	Reste de la division entière de Valeur1 par Valeur2.

Opérateurs logiques (modes entier seulement)

A	**And**	ET logique bit-à-bit de la *Valeur1* avec la *Valeur2*. Les bits du résultat sont positionnés à 1 s'ils sont également à 1 dans les positions correspondantes de *Valeur1* et *Valeur2*.
N	**Not**	Négation logique bit-à-bit de l'entrée actuelle.
O	**Or**	OU logique bit-à-bit de *Valeur1* avec *Valeur2*. Les bits du résultat sont positionnés à 1 s'ils sont à 1 dans les positions correspondantes de *Valeur1* ou de *Valeur2*.
X	**Xor**	OU exclusif bit-à-bit sur *Valeur1* avec *Valeur2*. Les bits du résultat sont positionnés à 1 si les bits correspondants dans Valeur1 ou Valeur2 sont positionnés à 1 mais pas les deux à la fois.
L	**shL**	Décalage à gauche de *Valeur1* de *Valeur2* bits. Si *Valeur2* est plus grand que 31 ou plus petit que 0, le résultat est égal à 0.
R	**shR**	Décalage à droite de *Valeur1* de *Valeur2* bits. Si *Valeur2* est plus grand que 31 ou plus petit que 0, le résultat est égal à 0.

Autres commandes

T **Affecte la valeur actuelle à une touche**

Convertit la valeur actuelle en une macro SMACS. (Cette commande suppose que SMACS est chargé). En mode réel, vous devrez préciser si vous voulez une chaîne de caractères sous la forme d'un tableau de constantes typées. Par exemple, si la valeur actuelle (ligne 2 de la fenêtre de calculs) était égale à Pi, la chaîne construite par CALC serait '3.14159265358979324' (format normal) ou '($40, $24, $93, $97, $58, $53, $26, $59, $41, $31)' (constante de type BCD). Cette commande est très pratique pour écrire des programmes utilisant l'unité TPBCD puisque le compilateur Turbo ne permet pas d'assigner une valeur constante à un réel BCD.

(ESC)	**Sortie**	Sortie de CALC.

(ALT)-U (ALT)-U Déchargement

Décharge le programme CALC de la mémoire, s'il est possible de le faire en toute sécurité.

Autres touches

(BACKSPACE) Efface le dernier chiffre ou caractère entré.

(F5), (F6), (F7), (F8), (F9), (F10)

En mode hexadécimal, ces touches peuvent être utilisées pour entrer des valeurs $A à $F respectivement.

(CTRL)-A, (CTRL)-B, (CTRL)-C, (CTRL)-D, (CTRL)-E, (CTRL)-F

En mode hexadécimal, ces touches peuvent être utilisées pour entrer des valeurs $A à $F.

. (point) Insère un point décimal (mode réel seulement).

E Dans les deux modes réels, **E** sert à marquer le début d'un exposant. Si la lettre **E** a déjà été utilisée une première fois pour cet objet, la frappe de **E** une seconde fois entraînera un effacement de l'entrée.

- (signe moins) Si aucun chiffre n'a été entré, le signe moins entraînera la négation des nombres qui seront entrés. Ce signe peut être également utilisé pour entrer un exposant négatif. Remarquez que les nombres négatifs ne peuvent pas être entrés dans les modes binaire ou hexadécimal ou lorsque l'opération actuelle entraîne le décalage de bits sur la gauche ou sur la droite.

+ (signe plus) Le signe + peut servir à donner le signe de l'exposant. Toutefois, la valeur prise par défaut est un signe positif.

Remarques sur les changements de modes de calculs

Bien qu'il soit normalement possible de passer du mode entier au mode réel et vice et versa, il y a quelques restrictions.

Premièrement, vous ne pouvez pas passer du mode entier au mode réel si l'opération actuelle n'est pas admise en mode réel (Mod, Or, Xor, etc).

Deuxièmement, vous ne pouvez pas passer du mode réel au mode entier si l'un des nombres dans la fenêtre de calcul est hors des limites de l'intervalle d'entiers (-2147483648 à 2147483647).

Finalement, lors du passage du mode réel au mode entier, les nombres fractionnaires sont toujours arrondis à la valeur entière la plus proche. Cela signifie que les nombres compris entre 0 et 1, tel que 0.25 seront convertis en 0 ou 1. Cela signifie également que les résultats calculés en mode réel apparaîtront souvent erronés après un passage en mode entier. Par exemple, si l'expression est "0.49 + 0.49 = 0.98" et que vous passiez en mode décimal, le calcul apparaîtra comme "0 + 0 = 1" (tout le monde sait cela...). Le fait de revenir au mode réel ne corrigera pas l'affichage.

GESTEMPS : Utilitaire de gestion d'emploi du temps

La plupart des programmeurs professionnels ont besoin de garder la trace du temps qu'ils ont passé sur chaque projet. Mais cela peut devenir difficile si vous avez plusieurs projets à gérer en même temps et surtout si vous êtes fréquemment interrompu. Si ce problème est le vôtre, et même s'il ne l'est pas, vous trouverez dans GESTEMPS un outil utile.

GESTEMPS permet de conserver des enregistrements précis sur votre emploi du temps. Vous pouvez activer ou désactiver rapidement et facilement une horloge, utiliser la pause pour prendre en compte les interruptions, sauvegarder et charger des fichiers de données, positionner une alarme, activer une horloge-écran, vous pouvez même planifier l'exécution d'une macro SMACS alors que vous êtes éloigné de votre bureau. Puisque GESTEMPS est un programme résident, vous n'aurez pas à interrompre votre travail pour l'utiliser.

Une fois que GESTEMPS est chargé en mémoire, vous pouvez l'activer n'importe quand en appuyant sur (CTRL)-(SHIFT)-T (shift gauche). En faisant cela, vous verrez apparaître l'écran ci-dessous :

```
================ GESTEMPS: Gestionnaire de temps 5.00 - AIEA ================
 Alarme:00:00    Macro:00:00    Début:03:39    Ecoulé: 00:00   Heure:03:39:48
 Fichier: C:\TURBO\PROJACT.DAT                 Minut: On  Pause: On  Hor: Off

 ESC: sortie  Alarme  Hor1  Suppr  Charge  Macro  Pause  minuTerie  miseàJour
=============================================================================
```

La ligne supérieure affiche de gauche à droite l'heure actuelle, l'heure à laquelle vous avez commencé votre travail, le temps écoulé depuis le début de votre travail (moins le temps pendant lequel l'option pause était active), l'heure à laquelle vous avez programmé une alarme et l'heure à laquelle une macro a été planifiée pour être exécutée. La deuxième ligne affiche le nom du fichier de données actuel, et l'état des options compteur de temps, pause, et horloge-écran. La dernière ligne affiche un menu des commandes disponibles décrites ci-après.

Alarme

En appuyant sur **A**, vous devrez répondre au message :

```
Entrez l'heure sous le format "HH:MM" (HH = 0-23, MM = 0-59):
```

Si vous voulez planifier une alarme pour 1 heure de l'après-midi par exemple, vous entrerez

```
13:00
```

Pour annuler l'alarme avant qu'elle soit active, appuyez simplement sur (ESC) au moment d'entrer l'heure. Si l'alarme a déjà été activée, elle peut être annulée ultérieurement en appuyant sur **A** une deuxième fois et en entrant **O** pour confirmer que vous voulez l'annuler.

Horl

L'option d'horloge-écran est activée si vous appuyez sur **H**. Lorsque vous quittez GESTEMPS, l'heure actuelle sera affichée approximativement trois fois par seconde. Pour désactiver l'horloge, lancez GESTEMPS et appuyez sur **H** une deuxième fois. Remarquez que l'horloge-écran est toujours désactivée temporairement lorsque GESTEMPS est activée.

Suppr

Cette option permet dans de nombreux cas de supprimer GESTEMPS de la mémoire sans avoir à relancer le système. Ceci pouvant être potentiellement dangereux puisque GESTEMPS ne pourra pas se désactiver lui-même si un programme qui a été chargé au-dessus de lui utilise le même vecteur d'interruption.

Charge

Vous devrez entrer le nom du fichier dans lequel vous voulez enregistrer les données lorsque vous appuyerez sur **C**. Si le fichier n'existe pas, un nouveau fichier de données sera créé. Sinon, GESTEMPS enregistrera simplement le nom du fichier pour les utilisations futures. Cette commande n'est pas disponible pendant que le compteur de temps est actif.

Macro

Cette option permet de planifier l'exécution d'une macro SMACS. Si SMACS n'est pas chargé lorsque vous appuyez sur **M**, un message d'erreur sera affiché. Sinon, vous devrez appuyer tout d'abord sur la touche correspondante à la macro, puis entrer l'heure à laquelle la macro devra être exécutée. Par exemple, si vous voulez exécuter la macro associée à (ALT)-**P** à 5:30, vous devrez appuyer sur (ALT)-**P** et entrer :

```
17:30
```

Comme avec la commande Alarme, vous pouvez annuler l'enregistrement de la macro avant qu'elle ait été planifiée en frappant sur (ESC) en réponse à la demande de l'heure. Vous pouvez aussi l'annuler ultérieurement en appuyant sur **M** une deuxième fois.

Pause

La commande Pause permet de prendre en compte les brèves interruptions dans votre travail. Si le compteur de temps est activé et que vous êtes interrompu, vous pouvez temporairement désactiver le compteur de temps en appuyant sur **P**. Bien que le compteur de temps reste actif, le temps écoulé sera inchangé tant que la touche Pause n'est pas pressée une deuxième fois.

minuTerie

C'est la commande la plus importante de GESTEMPS. Lorsque vous appuyez sur **T**, le programme active un compteur de temps pour garder la trace du temps écoulé et ajoute un nouvel enregistrement sur le fichier de données actuel. Si aucun fichier de données n'a été sélectionné avec la commande *Charge*, vous devrez préciser le nom d'un fichier. Pour désactiver le compteur de temps, appuyez sur **T** une deuxième fois ; le dernier enregistrement du fichier de données sera alors mis à jour.

miseàJour

La commande *miseàJour* existe principalement pour vous permettre de récupérer n'importe quelle erreur système qui pourrait se produire pendant l'activité du compteur de temps. Si cela se produit, vous pouvez simplement recharger GESTEMPS après avoir relancé le système, activez le programme, appuyez sur **J** pour *miseàJour* et entrez le nom du fichier de données. Lorsque GESTEMPS a mis à jour le dernier enregistrement sur le fichier, vous pouvez alors reprendre le comptage du temps en exécutant la commande *minuTerie*. La seule donnée que vous pourriez perdre pendant une erreur système est celle relative à la durée des pauses de votre travail. Dans ce cas, vous pouvez utiliser *miseàJour* à la fin de chaque pause. Remarquez que vous pouvez également récupérer des erreurs système manuellement, en modifiant le dernier enregistrement du fichier de données avec l'utilitaire GENETAT décrit par la suite.

GENETAT : Générateur d'états pour GESTEMPS

GENETAT est un utilitaire pour générer des états à partir du fichier de données produit par GESTEMPS. Il permet d'examiner, de modifier, d'ajouter, de supprimer des enregistrements dans le fichier de données ainsi que calculer des salaires si vous utilisez GESTEMPS en gestion de personnel.

Lorsque vous chargez GENETAT, le menu suivant s'affiche :

```
GENETAT: Générateur d'état pour les fichiers de données GESTEMPS 5.00
Ajoute Browse Calcule Détruit Modifie Fich Imp Quitte Etat Totaux
```

Les options qui suivent sont disponibles à partir de ce menu :

Ajout

Ajoute un nouvel enregistrement à la fin du fichier de données. Si un fichier a déjà été sélectionné, le nouvel enregistrement sera ajouté immédiatement à la fin de ce fichier. Vous aurez alors la possibilité de modifier l'enregistrement. Voir les explications de la commande *Edit*.

Browse

Balaie les enregistrements du fichier. Cette commande vous place dans le mode *Browse* dans lequel vous pouvez examiner tous les enregistrements d'un fichier de données. (PGUP) et (PGDN) permettent de parcourir des enregistrements en avant ou en arrière, (HOME) vous place sur le premier enregistrement du fichier et (END) sur le dernier enregistrement.

Calcul

Calcule les salaires à partir des heures travaillées. Si aucun fichier n'a été sélectionné, vous devrez en spécifier un. Vous devrez alors entrer un taux horaire pour le calcul des salaires. Pour annuler la commande, appuyez sur (ESC). Lorsqu'un taux horaire a été entré, les salaires seront calculés à partir du total cumulé des temps passés pour tous les enregistrements du fichier.

Détruit

Efface les enregistrements d'un fichier. La sélection de cette commande vous place dans le mode *Browse* (voir l'entrée de la commande *Browse*). Utilisez (PGUP) ou (PGDN) pour identifier l'enregistrement à supprimer, puis appuyez sur (ENTER). Puisque la suppression d'un enregistrement est irréversible, un message vous demandera de confirmer votre mise à jour. Vous pouvez poursuivre la suppression d'enregistrements de cette manière jusqu'à ce que tous les enregistrements soient supprimés ou que la touche (ESC) soit sélectionnée.

Edit

Modifie un enregistrement d'un fichier de données. La commande *Edit* vous place en mode *Browse* (voir l'entrée de la commande *Browse*). Utilisez (PGUP) ou (PGDN) pour localiser l'enregistrement à modifier puis appuyez sur (ENTER). Il y a 7 rubriques dans chaque enregistrement qui peuvent être modifiées. 6 d'entre elles vous sembleront probablement familières et se passeront d'explications : (1) la date de démarrage, (2) l'heure de début, (3) l'heure d'arrêt, (4) les arrêts, (5) le temps écoulé, (7) et un commentaire facultatif. Les instructions de base pour modifier ces rubriques sont affichées au bas de l'écran. Le sixième champ, Mise à jour autorisée, devrait toujours être à N(on). Cette rubrique indique au programme GESTEMPS si le dernier enregistrement doit être mis à jour ou non (voir les explications de GESTEMPS). GESTEMPS positionne cette rubrique à Oui ("True") lorsque l'enregistrement est créé et à Non ("False") lorsque le compteur de temps est désactivé. Après avoir modifié un champ, pressez (ENTER) pour vous placer sur le champ suivant. (ESC) annule toutes les modifications sur un enregistrement de base de données. Pour la liste complète des commandes d'édition, consultez la documentation de l'unité TPSAISIE.

Fichier

Sélectionne un nouveau fichier de données. Vous devrez entrer le nom d'un fichier de données GESTEMPS valide. Si le fichier est trouvé et qu'il est valide, toutes les commandes ultérieures opéreront sur ce nouveau fichier. Vous pouvez annuler cette commande en appuyant sur (ESC).

Imprim

Génère un état et l'envoie sur l'imprimante. Vous devrez entrer le nom du fichier de données si aucun fichier n'a déjà été sélectionné ainsi qu'un taux horaire. Si vous ne voulez pas spécifier un taux, appuyez sur (ESC). L'état envoyé à l'imprimante montrera chaque enregistrement du fichier, formaté de la façon suivante :

```
Enregistrement No 23
Début  :  1/25/89 à  9:47:30 am
Fin    :  1/25/89 à 10:21:21 am
Total  :  0:29:30        Gains : F17.21
```

La première ligne contient le numéro d'enregistrement. La deuxième ligne affiche l'heure de démarrage, et la troisième ligne affiche l'heure d'arrêt. La quatrième ligne affiche le temps écoulé, et si un salaire horaire a été spécifié, la somme gagnée. La cinquième ligne contient un commentaire, s'il existe. La fin de l'état affiche le temps total écoulé pour tous les enregistrements du fichier et le total des salaires (si un taux a été spécifié).

Quit

Quitte GENETAT. Si un fichier de données est ouvert, vous devrez confirmer votre intention d'arrêter.

Rapport

Génère un état et l'envoie sur un fichier disque. *Rapport* est semblable à *Imprim* mais la sortie est déroutée sur un fichier plutôt que sur l'imprimante. Le fichier créé est un fichier texte standard ASCII que vous pouvez modifier en utilisant un éditeur de texte.

Total

Affiche le nombre total d'enregistrements du fichier de données et les heures travaillées. Le total est semblable à *Calcul* mais il n'affiche pas les salaires.

GENETAT supporte également la souris, et utilise les possibilités de TPMENU, TPPICK et TPSAISIE. Voir la documentation de ces unités pour plus de détails.

REF : Table synoptique des caractères

REF est un programme résident en mémoire qui affiche les caractères utilisés par les programmeurs de PC. En reprenant l'idée des tableaux ASCII de SideKick, REF affiche un ensemble complet de tables de conversions pour les valeurs appartenant à l'intervalle 0-255. Ces tables

- représentent les caractères affichables
- affichent les codes ASCII
- affichent les valeurs en décimal, hexadécimal, binaire et octal
- affichent les touches et leurs valeurs associées, par exemple 16 = (CTRL)-P ou (ALT)-Q (code de balayage étendue) y compris les touches disponibles seulement sur les claviers étendus (affichées en haute intensité ou en blanc), aussi bien que les pseudo-codes de balayage renvoyés par certaines routines de l'unité TPSOURIS (affichés en haute intensité ou en jaune).
- affichent l'attribut vidéo correspondant à la valeur, à la fois pour les adaptateurs couleurs et monochrome (exemple couleurs fourni).

Pour lancer REF, appuyez sur (ALT)-(SHIFT)-R (shift gauche). Une fois que REF a été activée, les commandes suivantes sont disponibles :

(↑), (CTRL)-E, (CTRL)-W	Ligne précédente
(↓), (CTRL)-Z, (CTRL)-X	Ligne suivante
(PGUP), (CTRL)-R	Page précédente
(PGDN), (CTRL)-C	Page suivante
(CTRL)-(PGUP), (CTRL)-Q-R	Début du tableau
(CTRL)-(PGDN), (CTRL)-Q-C	Fin du tableau
(HOME), (CTRL)-Q-S	Début du tableau
(END), (CTRL)-Q-D	Bas du tableau
(ESC)	Sortie du programme
(ALT)-B	Bascule de l'attribut de clignotement. Si cet attribut de clignotement est désactivé dans le programme REF, il sera réactivé à la sortie du programme.
(ALT)-U-(ALT)-U	Déchargement de REF. Remarquez que REF, comme les autres programmes de démonstration résidents en mémoire, ne se décharge pas lui-même si cela n'est pas sûr. Si tel était le cas, REF retournerait à votre programme précédent, mais resterait résident.

Bien que l'affichage de REF soit auto-explicatif, certaines particularités demandent des explications, tels que les attributs vidéo.

Les attributs vidéo ont quatre composants comme le montre le schéma :

```
+---+---+---+---+---+---+---+---+
I 7 I 6 I 5 I 4 I 3 I 2 I 1 I 0 I    <-- Numéro des bits
+---+---+---+---+---+---+---+---+
  ^   ^   ^   ^   ^   ^   ^   ^
  I   I   I   I   I   I   I   I
  I   I   I   I   I   +---+---+--    Couleur des caractères
  I   I   I   I   +--------------    Bit d'intensité(affecte les caractères)
  I   +---+---+-----------------    Couleur du fond
+--------------------------------    Bits de clignotement
```

Comme vous pouvez le voir, tout attribut vidéo qui a une valeur supérieure à 127 (01111111 en binaire) produira un texte clignotant. Comme un texte clignotant peut être très désagréable s'il n'est pas utilisé avec modération REF supprime le clignotement dans tous ces exemples couleurs. Pour indiquer que les attributs produisent des clignotements, il met en valeur les noms des caractères et du fond. Si vous pressez (ALT)-B, le clignotement sera désactivé au niveau matériel, et le bit 7 (bit de clignotement), sera utilisé comme un bit d'intensité pour la couleur du fond. Cela permet d'obtenir plus de combinaisons de couleurs qu'en mode normal.

REF vous permet également d'utiliser une souris pour vous déplacer dans le tableau. Si une souris est installée, REF utilise l'extrémité gauche du tableau pour afficher un ascenseur. La position relative de l'attribut en haut de l'écran, dans le tableau, est représentée par un bloc d'une couleur différente. De chaque côté de ce bloc se trouvent des flèches, l'une pointée vers le haut, et l'autre pointée vers le bas. Pour dérouler légèrement l'affichage vers le bas, vous devrez placer le pointeur de la souris sur la flèche en bas et cliquer le bouton gauche de la souris. Pour dérouler vers le haut, déplacez le pointeur sur la flèche vers le haut, et cliquez le bouton gauche. Pour vous déplacez rapidement dans une autre zone du tableau, placez le pointeur de la souris sur l'un des blocs, et cliquez le bouton gauche. Vous obtenez la même chose en cliquant le bouton droit ou en pressant (ESC), quelle que soit la position du pointeur de la souris.

N'oubliez pas que pour utiliser la souris dans un programme résident, sans affecter les autres programmes, TPSOURIS (utilisée par REF), doit utiliser des fonctions de la souris qui ne sont pas implémentées dans tous les pilotes de souris (*MicroSoft* a ajouté ces fonctionnalités seulement pour leur utilisation dans *CodeView*). Malheureusement, il n'existe pas de moyens logiciels pour définir avec certitude si un pilote de souris donné, implémente ou non ces fonctions. Si vous rencontrez des problèmes en utilisant votre souris avec REF, vous devrez acquérir une version plus récente de votre pilote de souris, ou désactiver la directive *"UseMouse"* dans TPDEFINE.INC, et recompiler REF.PAS.

CREEMENU : Générateur de menus

CREEMENU est un générateur interactif de menus hiérarchisés. Il vous permet de visualiser vos menus pendant leur développement, de contrôler leur structure, de définir les couleurs et la présentation, beaucoup plus rapidement qu'avec *Turbo Pascal*. CREEMENU générera ensuite le code source Pascal ou créera un fichier binaire pour enregistrer un ou plusieurs menus.

Lorsque vous tapez CREEMENU à partir de la ligne de commandes du DOS, le menu d'accueil suivant s'affiche :

Aucun système de menus n'étant défini au démarrage, vous commencerez par en créer un. Aussi CREEMENU sélectionne automatiquement la commande Elément/Ajoute. Si vous pressez (ENTER), CREEMENU vous demandera le nom de l'élément à ajouter. Cependant, si vous utilisez CREEMENU pour la première fois, n'appuyez pas sur (ENTER), mais poursuivez votre lecture encore quelques minutes.

Lorsque vous parcourez les menus de CREEMENU, avec les touches curseurs, (ENTER) et (ESC), vous remarquerez qu'il vous est impossible de sélectionner certains éléments. Comme il n'existe pas de système de menus, certaines commandes sont désactivées puisqu'elles n'ont pas encore de rôle à jouer. Lorsque vous aurez créé au moins un élément dans votre menu, ou chargé un menu à partir d'une bibliothèque, CREEMENU activera le reste des commandes.

CREEMENU affiche toujours une ligne d'aide au bas de l'écran. Vous noterez également que de nombreuses commandes de CREEMENU, sont associées à des hot keys (affichées sur le côté droit des menus). Par exemple, vous pouvez accéder à la commande Global/Choix en pressant

(CTRL)-C. Dès que vous aurez utilisé CREEMENU, vous apprécierez l'efficacité des hotkeys pour les fonctions usuelles.

Si vous possédez une souris (compatible *MicroSoft*), vous pouvez l'utiliser avec CREEMENU. Utilisez le bouton gauche de la souris, au lieu de presser (ENTER) ; le bouton droit au lieu de presser (ESC), et les deux boutons à la place de la barre d'espacement mais seulement avec la commande Global/Choix. Vous pouvez déplacer la souris dans les propres menus de CREEMENU ou dans les menus que vous construirez.

Avant de décrire toutes les commandes de CREEMENU, nous allons construire un petit système de menus. Pour commencer, sélectionnez l'option Elément/Ajoute et pressez (ENTER). Entrez 'Donnée' comme nom d'élément suivi de (ENTER) (ne tapez pas les apostrophes). Vous pouvez alors saisir une ligne d'aide comme celle qui est affichée au bas de l'écran de CREEMENU. Pour l'instant, frappez (ENTER) sans saisir de ligne d'aide.

Vous verrez alors apparaître 'Donnée', dans un cadre situé dans le coin supérieur gauche de l'écran. Vous noterez également que toutes les commandes de CREEMENU sont maintenant activées, et que vous pouvez déplacer librement le curseur dans tous les sous-menus. Ajoutons un deuxième élément appelé 'Imprime'. Vérifiez que la barre lumineuse est positionnée sur Elément/Ajoute, et pressez (ENTER) à nouveau. Vous remarquerez que votre menu a grossi et que l'élément 'Imprime' est placé derrière l'élément 'Donnée' dans le cadre.

En réalité, vous voulez un menu qui se déroule horizontalement et non verticalement. Changeons alors l'orientation. Pour cela, pressez (←) pour vous placez sur la commande sous-menu de CREEMENU. Placez le curseur sur Elément et pressez (ENTER) pour sélectionner le menu suivant. Maintenant, placez le curseur sur Orientation et pressez (ENTER), puis sélectionnez Horizontal et pressez (ENTER). Votre menu apparaît maintenant horizontalement. Le cadre a été adapté et les éléments 'Donnée' et 'Imprime' apparaissent côte à côte.

Adaptons encore légèrement notre menu. Pressez (ESC) pour sélectionner sous-menu, et placez le curseur sur l'option Ombre et pressez (ENTER). Votre menu comporte maintenant une partie ombrée. Vous pouvez ainsi choisir n'importe quelle option pour adapter votre menu, et presser (ENTER) pour voir immédiatement apparaître les modifications du sous-menu courant, sans affecter les autres sous-menus.

Ajoutons maintenant un sous-menu attaché à l'élément 'Imprime'. Pour cela, pressez (ESC) pour sortir du menu élément et placez le curseur sur la commande Ajoute. Pressez (ENTER) pour ajouter un sous-menu imbriqué. Vous verrez immédiatement un petit cadre apparaître sous 'Imprime'.

Maintenant vous pouvez ajouter des éléments à ce nouveau sous-menu. Utilisons une hot key pour sélectionner la commande Elément/Ajoute. Pour cela, pressez (CTRL)-I. Les menus de CREEMENU ne changent pas, puisque nous voulons laisser le contexte actuel intacte. CREEMENU vous demande maintenant d'entrer le nom du nouvel élément. Entrer 'Options' et laissez à nouveau la ligne d'aide vide. Pressez (CTRL)-I pour ajouter un autre élément, et tapez 'Go' comme nom de ce nouvel élément.

Nous allons maintenant adapter le nouveau sous-menu. Placez la barre lumineuse sur la commande Elément à nouveau et pressez (ENTER) et sélectionnez la commande Cadre. Placez la barre lumineuse sur la commande n°4 de ce menu, pressez (ENTER) et maintenant votre sous-menu est rattaché correctement au menu parent.

Nous allons maintenant ajouter un sous-menu, à l'élément 'Donnée'. Pour cela, vous devez utiliser une autre commande de CREEMENU, la commande Choix. Pour activer la commande Choix, pressez (ESC) plusieurs fois jusqu'à ce que la ligne du menu principal de CREEMENU soit affichée, et pressez alors (←) pour vous placer sur la commande Global. Pressez (ENTER) pour dérouler le menu, puis (ENTER) à nouveau pour activer la commande Global/Choix.

Cette commande vous permet alors d'utiliser votre système de menus réellement. Vous pouvez utiliser les flèches, (ENTER) et (ESC) pour parcourir tout votre système de menu. Lorsque la barre lumineuse est placée sur un élément que vous voulez modifier, pressez la barre d'espacement pour rendre le contrôle à CREEMENU. Dans ce cas, vérifier que la barre lumineuse est sur 'Donnée' et pressez alors la barre d'espacement.

Activez alors la commande Sous-menu/Ajoute, et pressez (ENTER). Vous verrez alors apparaître un nouveau sous-menu sous l'élément Donnée. Répétez la même séquence d'opérations que vous avez utilisée pour créer ces éléments : pressez (CTRL)-I pour Elément/Ajoute, entrez 'Lecture', puis (CTRL)-I, et entrez 'Edit'. Ces éléments apparaîtront dans le sous-menu que vous venez d'ajouter. Maintenant, vous pouvez utiliser les commandes Sous-menu/Eléments/Cadre à nouveau pour ajuster le cadre de ce sous-menu avec le menu parent.

Notez que les sous-menus que nous venons d'ajouter, ne comportent pas de zones d'ombre même si le menu parent en comporte une. La commande que nous avons utilisée pour le menu parent ne concernait que ce seul menu. Pour modifier les valeurs par défaut des menus, vous devrez utiliser les commandes Global/Défaut de CREEMENU qui sont semblables aux commandes de Sous-menu/Eléments.

Pour terminer notre exemple, sauvegardons le menu sur disque, et générons également le code source approprié. Pressez (ESC) pour revenir au niveau de commandes primaires de CREEMENU puis (→), pour sélectionner l'option Fichier, puis (ENTER) pour sélectionner le sous-menu. Placez le curseur sur la commande *Ecrit Bibliotheque* et pressez (ENTER) . Entrez ensuite un nom de fichier qui n'existe pas. Souvenez-vous que CREEMENU ajoute automatiquement un suffixe par défaut égal à .LIB quel que soit le nom que vous entriez. Maintenant, CREEMENU vous demande d'entrer un identificateur de menus dans la bibliothèque. Ce nom de 12 caractères au plus permet de différencier les systèmes de menus qui sont enregistrés dans une même bibliothèque. Vous aurez besoin de cet identificateur lorsque vous rappelerez un système de menus à partir de la bibliothèque, aussi, choisissez un nom mnémonique. Dans ce cas, entrons 'TEST1'. CREEMENU écrit rapidement un fichier binaire de votre système de menus dans la bibliothèque, et vous demande de presser (ESC) pour continuer.

Puisque le menu a été écrit sur disque, CREEMENU désactive la commande *EcritBibliotheque* jusqu'à la prochaine modification du système de menu. Vous pouvez quoi qu'il en soit encore générer le code source Pascal pour votre système de menus, et CREEMENU positionne automatiquement le curseur sur cet élément. Pressez (ENTER) et CREEMENU vous demandera d'entrer un nom de fichier. Entrez le nom d'un fichier qui n'existe pas en vous souvenant que CREEMENU ajoute le suffixe .Pas par défaut, quel que soit le nom de fichier saisi, et qu'il écrasera tout fichier du même nom. Tapons par exemple, 'TEST1.PAS'. CREEMENU écrit le code source et vous demande de presser (ESC) pour continuer.

N'oubliez pas de sauvegarder votre menu dans la bibliothèque avant de quitter CREEMENU, si vous modifiez ce système de menus. La génération du code source n'est pas suffisante. CREEMENU peut lire un menu à partir d'une bibliothèque, de telle façon que vous puissiez le modifier à nouveau, mais il ne peut pas lire le code source. CREEMENU vous avertira toujours si vous essayez de quitter après avoir fait des modifications sans les avoir sauvegardées dans une bibliothèque.

Notre exercice est terminé. Vous pouvez maintenant sélectionner la commande Fichier/Quitte pour revenir au DOS. Vous pouvez compiler immédiatement TEST1.PAS, l'exécuter et jeter un autre regard sur le système de menus que vous venez de construire. Avec CREEMENU, vous serez émerveillé de la vitesse avec laquelle vous pouvez développer des systèmes de menus intégrés.

Les pages qui suivent vous serviront de guide de référence pour toutes les commandes de CREEMENU.

Global

Choix (CTRL)-C

Parcourt le menu et sélectionne l'élément à modifier. Utilisez les flèches, (ENTER) et (ESC), comme vous le feriez dans le programme final. Lorsque la barre lumineuse est sur un élément que vous voudriez modifier avec CREEMENU, pressez la barre d'espacement.

Nouveau (CTRL)-N

Crée un nouveau menu. Pensez à faire une sauvegarde du menu actuel en bibliothèque. N'utilisez pas cette commande lorsque vous commencerez à utiliser CREEMENU.

Défauts

La modification des valeurs par défaut a deux effets. Premièrement, le paramétrage que vous fournirez s'appliquera à toutes les opérations suivantes. Deuxièmement, lorsque vous modifiez un paramètre par défaut, la modification s'applique au système de menus courant, en commençant par le sous-menu courant et ses menus enfants.

Lorsque cela a un sens, CREEMENU affiche la valeur par défaut courante sous l'élément du menu. CREEMENU comporte un ensemble de valeurs par défaut. Lorsque vous lisez un menu à partir d'une bibliothèque, CREEMENU adapte les valeurs par défaut avec celles du menu racine de la bibliothèque.

Les éléments suivants apparaissent dans le sous-menu Défauts.

```
Couleurs
Cadre
Ligne d'aide
Orientation
Explosion
Ombre
Délai
adaPte
Restaure
```

Couleurs

Le sous-menu Couleurs contient les options : Cadre, Titre, Corps, Sélection, Choix, Aide, Désactive, Ombre. "Corps" est la couleur des éléments qui ne sont pas sélectionnés. "Choix" est la couleur de la lettre de sélection d'un élément qui permet de le sélectionner. "Sélection" est la couleur de chaque élément sélectionné. "Désactivé" est la couleur des éléments qui sont temporairement désactivés et qui ne peuvent être sélectionnés par aucun moyen. Lorsque vous sélectionnez l'une de ces couleurs pour la modifier, CREEMENU affiche une boîte de sélections de

couleurs. La valeur hexadécimale de l'attribut courant est affichée en haut de cette boîte. Utilisez les touches fléchées pour déplacer le losange en inversion vidéo, sur la nouvelle couleur et pressez (ENTER) pour la sélectionner. Pressez (ESC) pour quitter sans modifier la couleur.

Cadre

Type de cadre des sous-menus. Si le cadre que vous voulez n'existe pas, sélectionnez le plus proche et modifiez ultérieurement le code source.

Ligne d'aide

Numéro de ligne pour l'aide. Si vous ne désirez pas de ligne d'aide, entrez 0. Ceci désactivera la demande de ligne d'aide de la commande Elément/Ajoute.

Orientation

Permet de sélectionner des menus horizontaux ou verticaux.

Explosion

Permet de sélectionner des menus qui explosent ou non.

Ombre

Permet de sélectionner des menus ombrés ou non.

Délai

Délai en millisecondes entre deux explosions. Seule possibilité de modifier cette valeur puisque le délai entre deux explosions est une information globale de chaque système de menus.

Couleurs

Permet d'adapter ou non les couleurs. Lorsque cette option est activée, et que le système vidéo est monochrome (Mode Mono), ou composite noir et blanc (Mode Bw80), le système de menus adapte les couleurs pour améliorer la présentation avec ces modes.

Restaure

Restaure toutes les valeurs d'origine. Contrairement aux autres commandes globales par défaut, Restaure ne modifie pas immédiatement le système de menus en cours pour prendre en compte les nouvelles valeurs par défaut.

Redessine

Redessine l'écran s'il devient illisible. Cela devrait rarement arriver.

Info

Affiche l'espace disponible sur le tas pour CREEMENU.

Touches

Renumérote séquentiellement les touches de sélection des options pour le système de menus courant. Utilisez cette commande avec précaution si vous voulez modifier un système de menus que vous avez déjà intégré dans un programme, puisque les numéros de touches que vous avez déjà utilisés peuvent être modifiés.

Quelques mots de plus sur cette commande : chaque élément d'un système de menus doit avoir une touche de sélection de valeur unique. Habituellement, ces touches ont des valeurs petites que vous pouvez référencer dans une instruction CASE pour choisir le traitement à exécuter. CREEMENU affecte automatiquement les valeurs de touches à chaque élément que vous ajoutez en commençant à 1 pour le premier élément, et en utilisant le prochain entier disponible pour l'élément suivant. Si vous modifiez un système de menus ou si vous le créez au hasard, les numéros de touches ne suivront pas forcément une suite logique. C'est alors que la commande Global/Touches est pratique : elle affecte les numéros de touches du premier au dernier, en profondeur d'abord, ce qui génère les options en ordre numérique dans le code source. Il est recommandé d'utiliser la commande Global/Touches lorsque votre système de menus est terminé, juste avant de l'intégrer dans un programme.

Sous-menu

Ajoute (CTRL)-A

Ajoute un sous-menu sous l'option actuelle. Vous ne pouvez ajouter qu'un seul sous-menu à une option. Lorsque vous utilisez cette commande, CREEMENU affiche un petit cadre qui vous montre où le sous-menu sera placé. (Pour les sous-menus sans cadre, le cadre sera invisible sauf si la couleur de fond du corps n'est pas noire). Le sous-menu grossira automatiquement lorsque vous ajouterez des éléments ou lorsque vous utiliserez la commande Sous-menu/Redimensionne pour le modifier manuellement.

Entraîne (CTRL)-T

Modifie la position à l'écran d'un sous-menu et de ses descendants. Vous ne verrez pas les sous-menus descendants se déplacer mais lorsque vous les sélectionnerez à nouveau, vous verrez la différence. Utilisez la commande Sous-menu/Déplace pour déplacer un sous-menu sans repositionner ses descendants.

Eléments

Change les propriétés du sous-menu courant sans modifier les autres sous-menus ou les valeurs par défaut. Les options de la commande Eléments sont :

```
Couleurs
Cadre
Ligne d'aide
Orientation
Explosion
Ombre
```

Ces options se comportent comme celles de la commande Global/Défaut. Voir ci-dessus pour plus de détails.

Titre

Modifie le titre du menu. Le titre est affiché dans le haut du cadre du menu et n'apparaît seulement que lorsque le cadre est affiché. TPMENU tronquera le titre, s'il est plus long que la largeur du menu.

Déplace

Modifie la position à l'écran du seul sous-menu.

Redimensionne

Change la taille du sous-menu. CREEMENU recalcule automatiquement les dimensions d'un sous-menu lors de l'ajout d'un élément ou par exemple lorsque l'orientation du menu a été modifiée.

Supprime

Détruit tous les descendants de l'option en cours. Le sous-menu courant lui-même n'est pas détruit. Cette commande comporte un message de confirmation. Vous ne verrez pas immédiatement les effets de cette commande sur l'écran, puisque les menus imbriqués ne sont pas visibles à ce moment. Utilisez la commande Eléments/Supprime pour supprimer un élément de menu.

Eléments

Ajoute (CTRL)-I

Ajoute une option au sous-menu actuel. Vous devez ajouter chaque élément avant de pouvoir le modifier, le déplacer ou exécuter toute autre opération sur cet élément. Lorsque vous sélectionnez cette commande, CREEMENU vous demande le nom de l'élément. Puis, si l'aide est activée pour le système de menu, CREEMENU vous demande d'entrer la ligne d'aide à afficher lorsque l'élément est sélectionné.

Chaque élément est ajouté juste après l'élément en cours de sélection. Lorsque vous entrez un nouveau sous-menu, les éléments apparaissent les uns après les autres. Notez cependant que vous pouvez utiliser cette possibilité pour insérer de nouveaux éléments au milieu d'un sous-menu existant.

Edite (CTRL)-F

Edite le nom de l'option actuelle.

Déplace (CTRL)-D

Ajuste la position de l'option actuelle.

Notez que vous ne pouvez déplacer les éléments que dans une seule dimension. Pour les menus horizontaux, vous pouvez les déplacer de la gauche vers la droite, et pour les menus verticaux seulement de haut en bas. Notez également que le déplacement d'un élément ne modifie pas l'ordre dans lequel les éléments sont sélectionnés. Lorsque vous déplacez des éléments CREEMENU adapte les bords du menu.

Ordre (CTRL)-O

Ajuste l'ordre de l'option actuelle. Utilisez cette commande conjointement à la commande Elément/Déplace pour déplacer des éléments de menu et les re-cadrer.

Touche

Change la valeur numérique retournée quand l'option est sélectionnée. CREEMENU affiche une valeur entière par défaut à chaque élément que vous ajoutez. Si cette valeur est incorrecte, modifiez-la en utilisant cette commande. CREEMENU affiche toujours les valeurs de touches de l'élément courant à côté de cette commande.

Aide

Il permet d'entrer ou de modifier la ligne d'aide associée à l'élément de menu courant. La ligne d'aide sera affichée sur la ligne spécifiée par la ligne Sous-menu/Elément/Ligne d'aide pour le sous-menu courant. Si cette valeur vaut 0, la ligne d'aide ne sera pas affichée.

Car de sélection

Change le caractère de sélection de l'option actuelle. Les utilisateurs de votre système de menus peuvent presser le caractère de sélection et sélectionner rapidement l'élément correspondant d'un sous-menu. Les lettres de sélection de chaque élément devront donc être toutes différentes sinon vous désactiverez ce mode de sélection par caractères. Pour cela, pressez ← jusqu'à ce que la barre de sélection disparaisse. Sinon, utilisez les touches ← et → pour mettre en valeur le caractère que vous voulez sélectionner dans l'élément.

Supprime

Supprime l'option actuelle. La commande comporte un message de confirmation. Vous ne pouvez supprimer un élément tant que vous n'avez pas détruit tous les sous-menus descendants. Utilisez la commande Sous-menus/Supprime pour cela.

Fichier

Lit Bibliothèque CTRL-L

Lit un menu contenu dans une bibliothèque. Entrez le nom du fichier bibliothèque (qui comporte habituellement un suffixe .LIB), et l'identificateur du système de menus que vous voulez charger. Cet identificateur est une chaîne de 12 caractères au plus, que vous avez définie lors de l'écriture de la bibliothèque. Lorsque ces valeurs ont été saisies, CREEMENU lit et affiche le niveau primaire du système de menus sélectionné. Il adapte également les paramètres Global/Défaut pour qu'ils correspondent à ceux du niveau primaire du système de menus qu'il vient de charger.

Ecrit Bibliothèque CTRL-E

Ecrit un menu dans une bibliothèque. Entrez le nom du fichier bibliothèque. CREEMENU ajoute le suffixe .LIB par défaut, quel que soit le nom du fichier saisi. Entrez ensuite l'identificateur du menu, qui est une chaîne de 12 caractères au plus, pour différencier les systèmes de menus qui sont enregistrés dans le même fichier bibliothèque. Si l'identificateur que vous spécifiez existe déjà dans la bibliothèque, le menu courant remplacera celui de même nom. Puisque les fichiers bibliothèques de TPMENU contiennent au plus 10 systèmes de menus, et que les versions non utilisées de menus restent dans la bibliothèque tant qu'elle n'est pas compactée, il est possible que vous débordiez de l'espace alloué au répertoire de la bibliothèque si vous avez modifié le même système de menu plusieurs fois. Dans ce cas, utilisez la commande Fichier/Compacte bibliothèque pour supprimer les entrées inutilisées avant d'écrire une bibliothèque à nouveau.

Génère source (CTRL)-G

Génère le source Pascal correspondant au menu actuel. Entrez le nom du fichier dans lequel le code source doit être écrit. CREEMENU ajoute le suffixe .PAS par défaut. Le fichier source généré est prêt à être compilé et contient les instructions nécessaires pour que vous puissiez utiliser votre système de menus sans ajouter de lignes de programmation.

N'oubliez pas que CREEMENU ne peut pas lire directement et interpréter le code source. Si vous devez apporter des modifications à un système de menus, n'oubliez pas de le sauvegarder en format bibliothèque aussi bien qu'en code source.

Compacte Bibliothèque (CTRL)-B

Compacte une bibliothèque de menus en supprimant les menus obsolètes. Ceci est très important si vous avez enregistré un système de menus plus d'une fois en utilisant le même identificateur. Entrez le nom de la bibliothèque que vous voulez compacter. CREEMENU génère un fichier temporaire sur lequel il écrira le résultat. Si l'opération de compression se termine bien, CREEMENU supprime le fichier original et renomme le fichier temporaire avec le nom original.

Quitte (CTRL)-Q

Quitte CREEMENU et retourne au DOS. Si vous avez modifié votre système de menus depuis la dernière sauvegarde sur un fichier bibliothèque, vous pouvez encore changer d'avis.

BTRI : Utilitaire de tri de fichier

BTRI est un utilitaire rapide et puissant qui remplace la routine SORT du DOS. Utilisez-le pour trier des fichiers textes.

BTRI fonctionne en mode commande. La syntaxe est :

BTRI [Options] <FichierATrier >FichierTrié

Un écran d'aide est affiché. Si vous tapez simplement BTRI, les options sont :

```
/D    Tri en ordre décroissant
/I    Tri en ignorant  les majuscules/minuscules
/C n  Tri en commençant en colonne n
/M n  Tri sur un maximum de n caractères
```

Par défaut, le fichier est trié en ordre croissant et la longueur de la clé est égale à celle de chaque ligne. Les options peuvent être utilisées séparément ou combinées.

Exemples d'utilisation :

BTRI <TEST.DAT >TEST.TRI

Trie TEST.DAT en ordre croissant et écrit le résultat dans TEST.TRI

BTRI /I /D <TEST.DAT >TEST.SOR

Trie TEST.DAT en ordre décroissant en ignorant les majuscucles/minuscules

DIR | BTRI /C 10 /M 3

Trie un répertoire sur le suffixe.

BTRI fonctionne de deux manières. Si les options /C ou /M ne sont pas spécifiées, chaque ligne est stockée en mémoire pendant le tri et le résultat écrit directement sur le fichier de sortie. Sinon, seule la partie utile de chaque ligne est stockée en mémoire, la ligne concernée est alors recherchée sur le fichier d'entrée et recopiée entièrement sur le fichier résultat. Bien que la deuxième solution soit moins performante, elle permet d'économiser la mémoire pour trier de gros fichiers.

La longueur maximum d'une ligne est de 255 caractères. Vous pouvez utiliser TPASCIIZ pour traiter des lignes plus longues.

CHERCHE : Utilitaire de recherche dans des fichiers textes

CHERCHE est très pratique pour manipuler un grand nombre de fichiers. La recherche des fichiers sur disque s'effectue par nom ou par contenu.

CHERCHE est activé sur la ligne de commande. Si vous tapez simplement CHERCHE, un bref menu d'accueil s'affichera à l'écran et le programme s'arrêtera. La syntaxe complète est la suivante :

CHERCHE [Options] [ChaîneAChercher] [>FichSortie]

Le paramètre *ChaîneAChercher* spécifie la chaîne que vous voulez rechercher. Les options de recherche sont les suivantes :

```
/M              Recherche en respectant les majuscules et minuscules
/F              Affiche seulement les noms des fichiers
/E              N'exclut aucune extension de fichier dans la recherche
/R              Recherche récursive dans tous les sous-répertoires
/C[d:] [Chemin\] [masque]
                Commence la recherche dans d:chemin et ne recherche
                que dans les fichiers correspondant au masque
```

Si aucune option n'est fournie, CHERCHE traitera tous les fichiers concernés dans le répertoire courant. Puisque CHERCHE est conçu pour rechercher dans des fichiers textes, il exclut automatiquement les fichiers comportant certains suffixes : COM, EXE, OBJ, TPU, TPL, TPM, BIN, WKS. Vous pouvez ajouter d'autres suffixes à cette liste en modifiant une constante de CHERCHE.PAS et en recompilant le programme.

Toutes les fois que CHERCHE trouve une correspondance, il écrit le nom du fichier, le numéro de ligne et la ligne sur l'unité de sortie standard. Par défaut, CHERCHE ne tient pas compte des majuscules/minuscules.

Lorsque l'option /F est spécifiée, CHERCHE sélectionne chaque fichier contenant une chaîne appropriée. Il écrit le nom du fichier une seule fois sans numéro de ligne ou ligne de texte.

Lorsque l'option /E est spécifiée, CHERCHE traite également les fichiers binaires.

Lorsque l'option /R est spécifiée, CHERCHE traite non seulement le répertoire actuel mais aussi sur tous les sous-répertoires.

En utilisant l'option /C, CHERCHE traitera un répertoire différent du répertoire courant. Vous pouvez également spécifier les caractères génériques du DOS pour limiter la recherche à certaines catégories de fichiers.

Le paramètre *ChaîneAChercher* offre une souplesse considérable. Pour n'importe quel mot composé de caractères ASCII normaux, vous taperez simplement le mot. Si la chaîne à rechercher contient plusieurs mots séparés par des espaces, elle devra être mise entre guillemets ou apostrophes. Si la chaîne à rechercher contient des caractères de contrôle ou des caractères ASCII étendus, utilisez la notation suivante : #nnn où nnn est le rang du caractère ASCII.

Remarquez que la recherche est limitée à des lignes d'une longueur maximum de 1024 caractères.

Voici quelques exemples de commandes de CHERCHE :

CHERCHE /M /C *.PAS EcritVite

Recherche dans tous les fichiers .PAS du répertoire actuel les lignes contenant le mot EcritVite. La recherche se fait en tenant compte des majuscules/minuscules.

CHERCHE /R /C C:*.BAT >TOUTBAT

Donne la liste de tous les fichiers BAT dans chaque répertoire du lecteur C:. La liste sera enregistrée dans le fichier appelé TOUTBAT.

CHERCHE /F /E #129#14AuxSorties

Recherche tous les fichiers (y compris les fichiers binaires) dans le répertoire courant contenant la chaîne composée du caractère ASCII 129 suivi du caractère ASCII 14 suivi par le mot AuxSorties.

CHERCHE "Ne faites pas cela"

Recherche tous les fichiers textes du répertoire courant la phrase "Ne faites pas cela". Remarquez que des apostrophes peuvent être à l'intérieur d'une chaîne de recherche qui est délimitée par des guillemets et vice versa.

REPETE : Répétiteur de commandes

REPETE est un programme basé sur le programme REP des TurboPower Utilities. En tirant profit des bibliothèques *Turbo Professionnel*, REPETE est devenu plus compact et plus simple. Avec REPETE, vous pouvez automatiser des tâches fastidieuses et répétitives. Il lancera automatiquement un autre programme paramétré. REPETE est souvent utilisé, avec la sortie du répertoire ou le programme de recherche de fichiers pour exécuter une opération sur un groupe de fichiers.

REPETE est un programme activé par la ligne de commande. Si vous tapez simplement REPETE, un écran d'accueil s'affichera et le programme s'arrêtera. La syntaxe est la suivante :

REPETE "Commande" [Options] [<FichierEntrée]

Commande est le programme, le fichier batch ou la commande interne du DOS que vous voulez répéter. Si *Commande* contient plus d'un mot, il doit être placé entre guillemets. *Commande* ne peut pas spécifier directement une entrée ou une sortie redirigées, ou un piping. La présence de caractères <, > ou | entre guillemets peut induire le DOS en erreur qui les interprétera comme étant des re-directions de l'entrée ou de la sortie de REPETE au lieu de *Commande*. Les paramètres permettant de spécifier la re-direction de *Commande* sont présentés plus loin.

REPETE attend des données d'entrée qui lui indiquent comment personnaliser chaque répétition de *Commande*. Les données d'entrée sont lues à partir de l'unité standard d'entrée. Souvent les données de REPETE seront lues à partir d'une sortie obtenue par une liste de répertoire. Lorsque aucune donnée n'est spécifiée, REPETE répétera simplement la même commande une ou plusieurs fois. Ceci peut être utile en combinaison avec les autres options de REPETE décrites ci-dessous.

Options disponibles avec REPETE :

`/T ListeTouches`	`Passe les touches de ListeTouches à la commande`
`/Q`	`Demande de confirmation`
`/R n`	`Répète la commande n fois`

L'option /T demande à REPETE d'envoyer une suite de touches à l'application qu'il traite. Ceci est très pratique pour fournir certaines réponses telles que des réponses à la question "Etes-vous sûr ?". *ListeTouches* peut contenir du texte ou des caractères spécifiés par une suite numérique #nnn ou nnn est la valeur décimale ASCII du caractère désiré. Spécifiez les touches étendues avec #0 suivi par le code de balayage du caractère désiré. Par exemple, la touche F1 est écrite avec la suite #0#59.

Utilisez l'option /Q si vous voulez valider chaque commande de REPETE avant son exécution.

Utilisez l'option /R pour répéter simplement une commande donnée n fois. Cette option est valide seulement lorsque les données d'entrée ne sont pas fournies. Si des données d'entrée existent et que l'option /R n'est pas spécifiée, REPETE exécutera *Commande* une fois. La dernière possibilité sert lorsque REPETE est utilisé principalement pour passer des frappes de touches à une application.

Comment REPETE utilise-t-il les données d'entrée ? Il supporte ce que nous appelons des paramètres d'analyse. Chaque paramètre référencera une certaine partie de chaque ligne d'entrée et REPETE remplacera les paramètres toutes les fois qu'ils apparaîtront dans *Commande* ou dans *ListeTouches* avec les données réelles.

Liste des paramètres supportés par REP :

```
@n    Nième mot dans la ligne (n compris entre 1 et 9)
@O    Toute la ligne d'entrée
@B    Backslash du DOS
@D    Nom du répertoire correspondant au premier mot de la ligne
      d'entrée
@F    Nom de fichier correspondant au premier mot de la ligne d'entrée
@I    Symbole de re-direction de l'entrée <
@O    Symbole de re-direction de la sortie >
@P    Symbole |
```

REP partage chaque ligne d'entrée en mots séparés par des espaces ou des tabulations, comme la routine *ParamsStr* de *Turbo Pascal.* Vous pouvez référencer le premier mot avec @1 ou le neuvième mot avec @9. Spécifiez la ligne d'entrée tout entière avec le symbole @0.

Les symboles @B et @D et @F sont utiles pour analyser des noms de chemins. Ils fonctionnent très bien avec la sortie du programme CHERCHE fourni avec *Turbo Professionnel.* @D renvoie simplement la partie répertoire du chemin complet. @F renvoie simplement la partie nom de fichier. @B peut être utilisé pour recoller les deux parties. @B est remplacé par un backslash si la chaîne le précédant ne se termine pas par un backslash. Sinon, @B est remplacé par une chaîne vide.

Les symboles @I, @O et @P sont utilisés pour spécifier la re-direction de *Commande.* Ces symboles sont remplacés par les caractères standard du DOS avant l'exécution de *Commande.*

Quelques exemples pour vous permettre de mieux comprendre le fonctionnement de REPETE :

REP "DEL *.*" /T Y#13

Supprime tous les fichiers du répertoire actuel et répond automatiquement au message "Are You sure ?".

CHERCHE /R /C*.BAK | REPETE "DEL @0" /Q

Recherche tous les fichiers BAK dans tous les répertoires du disque actuel et les efface après confirmation.

DIR *.PAS | REPETE "DIFF @1.PAS @1.BAK @O@ODIFFS"

Recherche les différences entre chaque fichier Pascal du répertoire courant et le fichier de sauvegarde correspondant et place tous les résultats sur le fichier DIFFS.

CHERCHE /R /C *.* | REPETE "COPY @O A:\@F"

Recopie tous les fichiers du répertoire courant et de tous les sous-répertoires dans le répertoire racine du lecteur A:.

DIFF : Recherche des différences

DIFF compare deux fichiers textes et note leurs différences. C'est également un programme de démonstration des techniques de gestion mémoire du DOS : il supporte la mémoire normale, la mémoire EMS et la mémoire étendue.

DIFF est activé sur la ligne de commande. Si vous tapez DIFF seul, un écran d'accueil s'affichera et le programme s'arrêtera. Syntaxe des commandes DIFF :

DIFF FichierA FichierB [/N] [>Differences]

L'option /N de la ligne de commande force DIFF à utiliser la mémoire normale pour l'enregistrement du texte contrairement à la mémoire EMS ou la mémoire étendue. Par défaut, DIFF contrôle les trois sortes de mémoire et utilise celle qui dispose du plus grand espace pour enregistrer le texte pendant les opérations de comparaison (en fait, si la mémoire EMS et la mémoire étendue sont toutes les deux disponibles, il donnera la priorité à la mémoire étendue puisque l'accès y est plus facile).

DIFF écrit toutes les différences qu'il trouve sur l'unité standard de sortie. Le format est le suivant :

```
****Supprimer lignes 45-46 de T1
Donne une liste de tous les fichiers BAT de chaque répertoire du
lecteur C:.
La liste sera  enregistrée dans le fichier appelée TOUTBAK.
****Insérer lignes 45-46 à partir de T2
Recherche dans tous les fichiers (y compris les fichiers binaires) du
répertoire courant de la chaîne de caractères composée du code ASCII
129
```

Un commentaire explicatif commençant par '****' affiche le nom du fichier et les numéros de lignes dans lesquelles il y a une différence. Le commentaire explicatif est suivi du texte réel qui a été supprimé du fichier A ou inséré à partir du fichier B.

Remarquez que DIFF compare exactement les lignes. Les différences sur les majuscules/minuscules ou sur les espaces sont significatives.

VBROWSE : "Browse" de fichiers virtuels

VBROWSE utilise TPTBMVIR pour visualiser des fichiers de toutes tailles (en supposant que vous disposez d'assez de place sur votre disque). VBROWSE ne se mesure pas avec des utilitaires plus efficaces tels que LIST.COM. Il lit la tottalité du fichier en mémoire avant que vous puissiez le balayer et permet de créer de grands "fichiers d'échange". C'est pourquoi il est recommandé de posséder un disque dur avec suffisamment d'espace libre. L'objet du programme est principalement d'illustrer les routines de traitement des grands tableaux.

Appelez VBROWSE comme suit :

VBROWSE NomFichier [l:\chemin]

NomFichier est le nom du fichier à "browser" (VBROWSE recherche le fichier dans le chemin DOS éventuellement). Si aucun nom de fichier n'est mentionné, vous devrez en entrer un. "l:\chemin" est le nom d'un lecteur et d'un répertoire facultatifs pour le fichier d'échange TPTBMVIR. Cette option vous permet d'utiliser un autre lecteur (un disque virtuel) pour les fichiers temporaires. Par exemple :

VBROWSE wbrowse.pas l:

ordonne à VBROWSE de charger VBROWSE.PAS en mémoire et d'écrire son fichier d'échange sur l:.

Commandes disponibles avec VBROWSE

(←), (CTRL)-S	Déroule d'une colonne à gauche
(→), (CTRL)-D	Déroule d'un colonne à droite
(CTRL)-(←), (CTRL)-A	Déroule de 20 colonnes à gauche
(CTRL)-(→), (CTRL)-F	Déroule de 20 colonnes à droite
(↑), (CTRL)-E, (CTRL)-W	Déroule d'une ligne vers le haut
(↓), (CTRL)-Z, (CTRL)-X	Déroule d'une ligne vers le bas
(PGUP), (CTRL)-R	Déroule d'une page vers le haut
(PGDN), (CTRL)-C	Déroule d'une page vers le bas
(CTRL)-(PGUP), (CTRL)-Q-R, (HOME)	Début du fichier
(CTRL)-(PGDN), (CTRL)-Q-C, (END)	Fin du fichier
(CTRL)-Q-F	Recherche d'une chaîne de caractères

(CTRL)-L	Répétition de la dernière opération de recherche
(CTRL)-Q-L	Va à la ligne *L*
(CTRL)-K-n	Pose d'un marqueur *n* (0..9)
(CTRL)-Q-n	Va au marqueur *n* (0..9)
(ALT)-P, (CTRL)-K-P	Imprime les lignes
(ALT)-R	Active ou désactive le soulignement des lignes
(F3), (CTRL)-K-D, (ALT)-F	Charge un nouveau fichier
(ESC), (CTRL)-(BREAK)	Sortie du programme
(F1)	Affiche la liste des commandes disponibles

VBROWSE supporte également la souris (aussi bien que CREEMENU, GENETAT, REF et SAISIE). Les commandes de souris sont disponibles pendant le "browse" du fichier si la souris est installée.

Bouton droit	Sortie du programme
Les deux boutons	Affiche la liste des commandes disponibles

Les boutons de la souris ont les fonctions suivantes pendant la modification d'une chaîne de caractères avec l'éditeur ligne (de TPEDIT) ou lorsque vous devez presser une touche :

Bouton gauche	Identique à la frappe de (ENTER)
Bouton droit	Identique à la frappe de (ESC)

SAISIE : Exemple de saisie de données

Le programme de démonstration SAISIE.PAS mérite une certaine attention car il illustre les possibilités des nouvelles unités de *Turbo Professionnel* 5.0. Grâce aux unités TPDATE, TPSAISIE, TPPCIK et TPAIDE, il démontre par exemple comment concevoir des écrans de saisie de données sophistiqués avec contrôle des données ; comment implémenter des systèmes d'aide adaptés au contexte ; comment utiliser des listes pick ; comment lancer des tâches d'arrière-plan pendant l'attente d'une saisie clavier ; comment créer des commandes utilisateurs ; et comment écrire des programmes compatibles avec les différents formats de monnaie et de date.

Pour exécuter le programme de démonstration, vous aurez besoin de deux fichiers de la disquette DEMO : SAISIE.EXE et SAISIE.HLP (qui contient les messages d'aide). Au chargement du programe, l'écran est divisé en trois zones. la partie supérieure est divisée en deux parties, la deuxième partie sert de ligne d'état et affiche le numéro d'enregistrement courant, la date et l'heure. La date et l'heure sont actualisées par une routine définie par l'utilisateur est sont affichées comme une tâche d'arrière-plan. La partie centrale est l'écran de saisie de données composé de plusieurs champs que nous expliquerons plus loin. La partie inférieure est divisée aussi en deux parties. La première d'entre elles est une ligne d'aide qui fournit des instructions brèves sur la signification des zones de saisie. Le texte d'aide est modifié lorsque vous parcourez les différentes rubriques. La seconde partie est une ligne d'aide statique relative aux touches affectées aux fonctions de base.

L'écran de saisie principal, conçu avec TPSAISIE, comporte quinze rubriques. La liste qui suit décrit le rôle de chacune de ces rubriques :

Nom	Rien de très particulier sur cette rubrique. Si la rubrique contient déjà un texte et que le première touche pressée est un caractère ASCII, la rubrique sera effacée. Si l'éditeur de rubriques est en mode insertion, la fin de la rubrique peut être décalée. Pressez (ENTER) ou (TAB) pour vous placer sur la prochaine rubrique lorsque vous aurez terminé.
Adresse	Ce champ est obligatoire et ne peut être laissé vide
Ville	Contrairement à la rubrique Nom, le texte en fin de rubrique ne peut être décalé en mode insertion

Département	Vous avez deux possibilités. Vous pouvez entrer pour un nom de département abrégé (deux lettres) ou vous pouvez presser (F2) et sélectionner une abbréviation dans une liste pick créée avec TPPICK. Seuls les caractères numériques sont permis et un message d'erreur s'affichera si vous ne saisissez qu'une seule lettre.
Code postal	Vous devez entrer un nombre sur 5 positions. Un message d'erreur s'affichera si vous essayez d'entrer un code postal incorrect.
Téléphones	C'est un champ fictif qui génère un deuxième écran de saisie. Pour obtenir le deuxième écran affiché dans une fenêtre popup, pressez (ENTER). Les deux rubriques de ce nouvel écran sont expliquées ci-dessous.
Sexe	Rubrique à choix multiples. Pour sélectionner l'une des valeurs (Inconnu, Masculin, Féminin), pressez (_____) ou + pour dérouler la liste des choix en avant, ou - pour la dérouler en arrière.
Marié(s)	Rubrique de valeur Oui ou Non. Pressez **O** ou **N** ou (ENTER) pour valider la valeur par défaut.
Né(e) le	Rubrique de type date. Vous pouvez entrer n'importe quelle date entre 1/1/00 (1900) et 12/31/99 (1999). Les routines de contrôle de TPDATE et de TPSAISIE assurent la cohérence de la date saisie et un message d'erreur s'affiche si vous essayez d'entrer une date non valide (1/32/88).
Age	Champ calculé que vous ne pouvez modifier. Grâce à TPSAISIE, cette valeur est calculée et affichée dès que vous saisissez la rubrique date de naissance. Cette rubrique est cadrée à droite.
Salaire horaire	Rubrique numérique qui utilise l'éditeur numérique fourni par TPSAISIE. Les nombres sont entrés de la droite vers la gauche en mode calculette. Le point décimal est un point fixe et le signe F (dollar) un signe flottant.

Salaire hebdomadaire

Comme Age, il s'agit d'un champ calculé, sa valeur est égale à Salaire horaire * Nombre d'heures par semaine. Le point décimal est fixe et le signe F flottant. La rubrique est complétée avec des astérisques plutôt qu'avec des espaces.

Heures / semaine

Il s'agit d'un type spécial de rubrique à choix multiples. Plutôt que de dérouler une liste de choix, incrémentez ou décrémentez la valeur par défaut en utilisant les touches + et -.

Salaire annuel

Il s'agit encore d'un champ calculé. Sa valeur est égale à Salaire mensuel * 52. Le point décimal est fixe et le signe $ flottant. La rubrique est complétée avec des astérisques plutôt qu'avec des espaces.

Remarques

Cette rubrique a deux caractéristiques spéciales. Le curseur n'avance pas automatiquement sur la prochaine rubrique lorsque la fin de la rubrique est atteinte et la chaîne de caractères entrée sera aussi longue que la rubrique modifiée. La rubrique est déroulée horizontalement éventuellement.

Le deuxième écran de saisie, accessible en déplaçant le curseur sur la rubrique Téléphones et en pressant (ENTER) comporte deux rubriques :

Téléphone du bureau

Seuls les nombres et les espaces peuvent être saisis dans cette rubrique.

Téléphone personnel

Cette rubrique a les mêmes caractéristiques que la rubrique Téléphone du bureau.

SAISIE mérite aussi d'être noté pour son système d'aide adpatée au contexte, développé avec TPAIDE. Lorsque vous déplacez le curseur d'une rubrique à une autre, essayez de presser le touche (F1) pour obtenir des informations sur cette rubrique particulière. Un mot ou une phrase mis en valeur avec une couleur violette (sur les écrans couleurs) ou en inversion-vidéo (sur les écrans monochromes) correspond à un mot de liaison avec d'autres informations. La frappe de (ENTER) entraîne l'affichage de l'écran d'aide relatif à ce thème. Vous pouvez alors revenir au thème précédent en pressant (ALT)-(F1). Vous remarquerez que certains écrans d'aide ont plus d'un lien : si un mot ou une phrase est affiché en bleu clair (sur les écrans couleur) ou souligné (avec les écrans monochromes), vous pouvez sélectionner ce thème en plaçant le curseur sur ce mot et en pressant (ENTER). L'écran d'aide pour la rubrique Salaire horaire, par exemple, comporte quatre liens.

Pour voir l'index de tous les thèmes d'aide, pressez (F1) à partir du système d'aide. Vous pouvez alors sélectionner n'importe quel thème en déplaçant la barre lumineuse sur le thème approprié et en pressant (ENTER). Certains thèmes (ceux du début de la liste) sont inaccessibles depuis l'index. Ces thèmes concernent les commandes de déplacement du curseur et d'édition disponibles.

Bien que le système d'aide décrive brièvement chacune des commandes, lisez bien ce qui suit :

(ALT)-B
Normalement lorsque vous saisissez un caractère non valide, SAISIE émettra un bip. Pressez (ALT)-B pour activer ou désactiver le bip. C'est un autre exemple de commande définie par l'utilisateur.

(TAB), (SHIFT)-(TAB)
Ces touches placent le curseur sur la rubrique précédente ou le rubrique suivante. La rubrique Date de naissance par exemple comporte trois sous champ : Mois, Jour et Année.

(↑), (↓)
Ces touches déplacent le curseur vers le haut ou vers le bas sur l'écran mais pas sur la rubrique précédente ou suivante.

(PGUP), (PGDN)
Touches de déplacement vers l'enregistrement précédent ou l'enregistrement suivant. Bien que vous puissez saisir au plus 20 enregistrements, souvenez-vous que SAISIE ne fournit aucun moyen de sauvegarder ces enregistrements. Ne perdez donc pas trop de temps à remplir les différents enregistrements.

(ESC), (CTRL)-(BREAK)
Restaure l'enregistrement actuel à son ancien contenu et vous permet d'arrêter la session.

(CTRL)-(ENTER), (CTRL)-K-D, (CTRL)-K-Q
Fin de la session de saisie de données. L'enregistrement courant n'est pas restauré à sa valeur précédente comme lorsque vous pressez (ESC).

SAISIE supporte la souris que vous pouvez utiliser pour :

- Placer le pointeur de la souris (en forme de losange) sur un champ non protégé de l'écran de saisie et le sélectionner en cliquant le bouton gauche de la souris
- Placer le pointeur de la souris sur le champ Téléphone et cliquer le bouton gauche de la souris : le deuxième écran de saisie sera activé.
- Placer le pointeur de la souris sur le champ Département et cliquez le bouton gauche de la souris une première fois pour le sélectionner et une deuxième fois pour afficher la liste pick des noms de départements.
- Placer le pointeur de la souris sur le champ Marié(e) et cliquez le bouton gauche de la souris une première fois pour le sélectionner et une deuxième fois pour changer la valeur.
- Placer le pointeur de la souris sur le champ Heures/semaine et cliquer le bouton gauche de la souris une première fois pour le sélectionner et une deuxième fois pour incrémenter la valeur.
- Presser le bouton droit de la souris à la place de (ESC)
- Presser les deux boutons (gauche et droit) pour afficher l'écran d'aide du champ courant.
- Presser le bouton central de la souris (sur les souris à trois boutons seulement) à la place de (ENTER).

Vous pouvez aussi utiliser la souris pour sélectionner des élément dans une liste pick (voir TPPICK pour plus de détails), pour utiliser le système d'aide (voir TPAIDE) et pour répondre à des demandes par de simples touches (en général, cliquez le bouton droit est équivalent à presser (ESC), alors que cliquer le bouton gauche est équivalent à pressez (ENTER)).

Autres programmes

Contrairement à la plupart des programmes de démonstration qui sont livrés avec *Turbo Professionnel*, qui sont destinés également à l'apprentissage, trois d'entre eux sont de véritables attractions. EXPLFENT met en oeuvre certains effets visuels étonnants qui peuvent être créés en utilisant des écrans virtuels gérés par l'unité TPECRAN, alors que DEROULFE illustre les possibilités du gestionnaire de fenêtres popup de TPFENETR qui met en oeuvre un ensemble de routines pour travailler avec des écrans virtuels.

EXPLFENT

EXPLFENT gère cinq écrans différents (quatre d'entre eux sont des écrans virtuels) et crée certains effets attractifs en fusionnant des écrans virtuels avec l'écran physique de différentes façons. Des petites fenêtres sont placées sur l'écran instantanément et la dernière est alors déplacée sur l'écran en donnant l'illusion qu'elle efface les autres. Appuyez sur (ESC) pour arrêter le mouvement incontrôlé de la fenêtre et revenir au DOS.

DEROULFE

DEROULE crée tout d'abord 8 fenêtres popup de tailles et de couleurs aléatoires, puis les déplace sur l'écran à une vitesse fantastique, en les déroulant, en les agrandissant et en les réduisant. Pressez une touche pour reprendre le contrôle et revenir sous DOS.

14 Annexes

Ordre d'écriture des unités

La liste qui suit donne l'ordre d'écriture de toutes les unités de la bibliothèque *Turbo Professionnel* lorsqu'elles sont utilisées ensemble. Les unités qui peuvent être placées en "overlays" en utilisant *Turbo Pascal* 5.0 sont suivies d'un astérisque (*)

```
TP8087
   Pas d'ordre

TPASCIIZ*
   TPCHAINE

TPBCD*
   Pas d'ordre

TPCRT
   TPINLINE

TPDATE*
   DOS
   TPINLINE

TPDIR*
   DOS
   TPCRT
   TPDATE
   TPDOS
   TPINLINE
   TPPICK
   TPCHAINE
   TPFENETR

TPDOS*
   DOS
   TPCHAINE
```

TPTBMEXP*
   DOS
   TPEMS
   TPMEMCHK

TPEDIT*
   TPCMD
   TPCRT
   TPCHAINE
   TPSOURIS

TPEMS
   Pas d'ordre

TPCLPLUS
   Pas d'ordre

TPSAISIE*
   TPBCD
   TPCMD
   TPCRT
   TPDATE
   TPINLINE
   TPMEMCHK
   TPSOURIS
   TPCHAINE

TPEREXEC
   DOS

TPMEMEXT*
   DOS

TPAIDE*
   DOS
   TPCMD
   TPCRT
   TPDOS
   TPMEMCHK
   TPSOURIS
   TPPICK
   TPCHAINE
   TPFENETR

TPINLINE
   Pas d'ordre

TPINT
   DOS

TPINT24
   Pas d'ordre

TPEDMAC*
- DOS
- TPCRT
- TPINT
- TPMACRO

TPMACRO
- DOS
- TPINT

TPMENU*
- TPCMD
- TPCRT
- TPMEMCHK
- TPSOURIS
- TPCHAINE
- TPFENETR

TPSOURIS
- TPCRT

TPPICK*
- TPCMD
- TPCRT
- TPSOURIS
- TPCHAINE
- TPFENETR

TPTBMRAM*
- TPINLINE
- TPMEMCHK

TPECRAN
- DOS
- TPCRT

TPTRI*
- Pas d'ordre

TPCHNFOR
- Pas d'ordre

TPCHAINE
- Pas d'ordre

TPTSR
- DOS
- TPINT

TPTBMVIR*
- DOS
- TPMEMCHK

TPFENETR*
- TPCRT
- TPMEMCHK

Les nouveautés de Turbo Professionnel 5.0

Cette partie présente brièvement les améliorations de la version 5.0. Elle est destinée principalement aux utilisateurs de la version *Turbo Professionnel* 4.0.

La version *Turbo Professionnel* 5.0 a augmenté d'environ 50 %. Il y a maintenant à peu près 600 routines au lieu de 400 et 31 unités au lieu de 25 ce qui nous a conduit à rassembler la documentation en deux volumes.

Les unités ont été modifiées pour être compilées à la fois avec *Turbo Pascal* 4.0 et 5.0. Lorsque cela était nécessaire, nous avons ajouté le support de nouvelles fonctionnalités de Turbo 5.0. TPTSR vous permet d'écrire des programmes résidents avec overlays ou qui utilisent l'émulation "virgule flottante". TP8087 et TPEREXEC supportent maintenant cet émulateur. Nous avons également ajouté les directives de compilation pour certaines unités overlays de *Turbo Professionnel* (voir appendice A).

Afin de gagner de la place en bibliothèque, nous avons abandonné la disquette TPU (qui contenait les fichiers TPU précompilés) et la disquette BONUS (ajoutée en version 4.03) et nous les avons converties en "Disquettes SRC" (comme dans code SouRCe). Comme nous devons gérer deux versions de compilateur, cela évite d'envoyer deux ensembles de fichiers TPU, qui conduirait à un gaspillage de disquettes.

Vous devez en contre-partie compiler les unités avant de les utiliser (cette contrainte est largement facilitée par le nouveau programme d'installation INSTALL.EXE et le fichier Make TPRO.MAK). Un répertoire BONUS a été ajouté sur la disquette SRC3 (il contient les utilitaires du domaine public).

L'augmentation de la taille du logiciel est due à l'ajout des nouvelles unités :

- TPAIDE : Routines d'aide
- TPSOURIS : Routines de gestion de la souris
- TPPICK : Listes pick
- TPDIR : Répertoire popup
- TPSAISIE : Ecrans de saisie de données
- TPDATE : Gestion de la date et de l'heure
- TPCLPLUS : Interface de clavier étendu

Les premières versions de certaines de ces unités étaient disponibles sur l'ancienne disquette BONUS, nous les avons adaptées et améliorées. Les unités TPSOURIS et TPSAISIE sont totalement nouvelles. TPMEMCHK et TPCMD sont réservées à l'usage interne des autres unités.

Nous avons aussi amélioré certaines unités existantes, TPCRT contient maintenant des routines pour sauvegarder et restaurer des écrans en format compacté ainsi que de nouvelles routines d'écriture sur l'écran et notamment la procédure *EcritFacile*. TPFENETR permet maintenant de créer des fenêtres ombrées qui explosent avec ou sans effet sonore et offre également plusieurs routines pour gérer des écrans virtuels. Grâce aux possibilités de TPFENETR, TPMENU permet maintenant de créer des menus ombrés qui explosent et de désactiver temporairement certains éléments d'un menu. TPINLINE.INC a été complétée et convertie en une unité (TPINLINE). TPEDIT est maintenant plus facile à personnaliser et à utiliser avec des écrans de saisie de données et autorise l'entrée de mot de passe. D'autres unités ont été sensiblement améliorées.

Dans certaines unités et notamment TPFENETR et TPMENU, ces améliorations nous ont conduit à modifier des structures de données internes. Ceci vous obligera éventuellement à modifier le code existant afin qu'il puisse être compilé correctement sous *Turbo Professionnel* 5.0. De tels cas seront exceptionnels puisque les structures de données modifiées n'étaient pas documentées dans la première version et que nous vous avons dissuader d'écrire des programmes qui utilisent nos structures internes de données. Dans tous les cas, comparez l'ancienne et la nouvelle version d'une unité avant de modifier votre code.

Nous avons également fait notre possible pour que la souris soit supportée dans toute la bibliothèque. TPEDIT, TPMENU, TPPICK, TPAIDE et TPSAISIE vous permettent d'exécuter des commandes de base qui utilisent la souris (Si vous n'avez pas besoin de la souris, vous pouvez la désactiver dans toutes les unités et programmes de démonstration grâce à la directive de compilation conditionnelle "UseMouse" de TPDEFINE.INC et en recompilant les unités ou les programmes).

Des modifications ont été apportées aux programmes de démonstration pour tenir compte des améliorations de la bibliothèque. La cas le plus significatif est GENETAT.PAS qui utilise maintenant TPSAISIE, TPDATE, TPDIR et TPSOURIS. Un nouveau programme SAISIE.PAS a été ajouté pour illustrer l'utilisation de plusieurs nouvelles unités : TPSOURIS, TPSAISIE, TPDATE, TPPCIK et TPAIDE. Plusieurs programmes de démonstration comportent maintenant un support de souris, notamment CREEMENU et REF.

Certains éléments de la version 4.0 ont été abandonnés notamment l'unité TPANCFIC. L'unité TPLIGCMD est encore distribuée avec le logiciel mais n'est plus documentée. Le programme de démonstration PILEFENT a été abandonné afin de gagner de la place.

Ce sont les modifications les plus significatives apportées au logiciel mais si vous parcourez rapidement la table des matières de la documentation vous y trouverez surement d'autres nouveautés. Prenez soin de relire les généralités au début de chaque chapitre important. Dans quelques cas, la documentation n'a pas changée mais dans d'autres, elle contient des informations nouvelles et importantes relatives à l'utilisation d'une unité donnée.

Gestion du clavier avec Turbo Professionnel

Comme vous avez pu le remarquer, plusieurs unités de Turbo Professionnel offrent des interfaces qui permettent au programmeur de contrôler le gestion du clavier, comme dans TPEDIT, TPSAISIE, TPPICK, TPEDMAC, TPMENU et TPAIDE. Plus précisément, il existe trois sortes d'interfaces qui permettent de gérer le clavier.

La première est *l'interface clavier programmable.* Cinq de ces six unités (toutes sauf TPEDMAC) définissent le jeu de commandes affectées à certaines touches ; elles enregistrent toutes les correspondances entre les commandes et les touches dans un tableau qui peut être facilement modifié. La deuxième unité est *l'interface avec les tâches d'arrière-plan.* Les six unités permettent au programmeurs de spécifier une fonction qui renvoie une touche ; cette fonction exécute la tâche que vous souhaitez pendant l'attente de la frappe d'une touche. La troisième est *l'interface d'aide adaptée au contexte.* Ces interfaces facilitent la création de systèmes d'aide adaptée au contexte en fournissant le moyen d'appeler une routine d'aide définie par le programmeur lorsqu'une commande particulière est exécutée.

Comme exemple d'utilisation de ces interfaces, essayez de compiler et d'exécuter le programme de démonstration SAISIE.PAS. *L'interface clavier programmable* permet au programme SAISIE de définir des nouvelles commandes qui peuvent être exécutées pendant la saisie de données (essayez de déplacer le curseur sur le champ "Code postal" et pressez A, vous entendrez un bip puis pressez (ALT)-B et essayez à nouveau, vous n'entendrez plus rien). L'interface de tâches d'arrière-plan permet au programme SAISIE d'afficher une horloge sur l'écran pendant la saisie de données à partir du clavier, et l'interface d'aide adaptée au contexte permet

d'activer les écrans d'aide pendant la saisie de données et pendant la sélection d'un nom de département dans une liste pick (déplacez le curseur sur le champ "Département" et pressez (F2) suivie de (F1)). L'utilisation de ces interfaces est réellement très facile.

Interface de clavier programmable

TPEDIT, TPSAISIE, TPPICK, TPMENU et TPAIDE utilisent toutes un tableau de constantes typées pour enregistrer les données qui relient une touche donnée ou une suite de touches à une commande. Bien que ces tableaux soient de tailles différentes, ils sont tous de type *ModifJeuTouches* de TPEDIT :

```
EditeMaxTouche = 150;
EditeIDTouche : string[16] = 'tpedit key array';
ModifJeuTouches : array[0..EditeMaxTouche] of byte = (
{Taille  Touches    Type commande      Séquence touches}
   3,    $00,  $00,        LCquitte,        {^Break}
   3,    $00,  $3B,        LCaide,          {F1}
   3,    $00,  $47,        LChome,          {Home}
   3,    $00,  $4B,        LCgauche,        {Gauche}
   3,    $00,  $4D,        LCdroite,        {Droite}
   3,    $00,  $4F,        LCend,           {End}
   3,    $00,  $52,        LCins,           {Ins}
   3,    $00,  $53,        LCdel,           {Del}
   3,    $00,  $73,        LCmotGauche,     {^Gauche}
```

EditeIDTouche est un identificateur utilisé par les programmes d'installation. *ModifJeuTouches* est une suite d'enregistrements de longueur variable composés de trois rubriques. La première contient la longueur des deux autres rubriques, la deuxième contient le code de la touche et la troisième contient la valeur associée à une commande donnée.

Le champ relatif aux codes de touches est de longueur variable. Dans la plupart des cas, les commandes sont affectées à une seule touche telle que (CTRL)-A et (HOME) ou à une séquence de deux touches telle que (CTRL)-Q-D et occupent donc un ou deux octets. Les caractères de contrôle occupent un octet alors que les touches étendues (touches de fonction, touches du curseur, etc) occupent deux octets : zéro suivi du code de balayage. Le nombre d'octets d'une touche est donc compris entre 1 (touche simple) et 4 (combinaison de deux touches étendues telle que (ALT)-D, (ALT)-R). Bien qu'il soit possible d'affecter une combinaison de trois touches ou plus à une seule commande, cela n'est par recommandé.

Il y a donc trois façons d'adapter les rôles des touches par défaut. La première et la plus évidente est de modifier simplement le tableau des touches dans l'unité en question de *Turbo Professionnel*. C'est le meilleur moyen pour que ces changements soient permanents et effectifs dans tous vos programmes. Néanmoins, nous vous recommandons vivement de bien

modifier les cinq unités (TPEDIT, TPSAISIE, TPPICK, TPMENU et TPAIDE) et d'étudier les affectations des touches par défaut. Les affectations de touches usuelles garantissent la cohérence des unités qu'il faudra préserver avant de modifier.

La seconde méthode correspond à des modifications mineures pour un programme donné et utilise l'une des routines suivantes :

```
AjoutCommandeEdit          {TPEDIT}
AjoutCommandeSaisie        {TPSAISIE}
AjoutCommandePick          {TPPICK}
AjoutCommandeAide          {TPAIDE}
AjoutCommandeMenu          {TPMENU
```

Toutes ces routines sont fondamentalement identiques à la fonction *AjoutCommandeEdit* de TPEDIT :

```
function AjoutCommandeEdit (
   Cmd : LCtype;
   NbTouches : bytes;
   Tche1, Tche2 : word) : boolean;
```

Ces routines vous permettent toutes de modifier les affectations des touches pour une unité. Elles vous permettent premièrement de définir de nouvelles commandes (que vous devrez traiter vous-même), deuxièmement de désactiver une commande donnée et troisièmement de modifier une affectation de touche. Nous allons vous donner un exemple de chaque possibilité.

Supposons que vous utilisiez TPEDIT et que vous vouliez permettre à vos utilisateurs d'utiliser ↑ et ↓ pour signaler que le curseur doit être placé sur une autre rubrique. Voici ce qu'il faudra faire :

```
{Affecte la première commande définie par l'utilisateur à <Up>}
if not AjoutCommandeEdit(LCutl0, 1, $4800, $0000) then
   {Erreur : le tableau des touches est déjà plein} ;
{Affecte la deuxième commande définie par l'utilisateur à <Down>}
if not AjoutCommandeEdit(LCutl1, 1, $5000, $0000) then
   {Erreur : le tableau des touches est déjà plein} ;
```

Le premier paramètre est le code commande. TPEDIT définit trente commandes (*LCRien..LCUlt9*) et les dix dernières (*LCUtl0..LCUtl9*) servent à définir des commandes utilisateurs. Dans cet exemple, nous utiliserons les deux premières pour ↑ et ↓. Le second paramètre indique si la commande est affectée à une seule touche (telle que ↑) ou à une suite de deux touches (telles que CTRL-**Q-D**). Les troisième et quatrième paramètres fournissent les codes de balayage des touches. Dans notre exemple, les commandes sont affectées à des touches simples, aussi seul le premier paramètre est significatif. Si nous ajoutons une commande *WordStar* par exemple CTRL-**Q-D**, nous passerons en paramètre *'Ord(^Q)'* (touche 1) et

'Ord(^D)' (touche 2). Notez que nous passons '^D' plutôt que 'D' : TPEDIT acceptera alors (CTRL)-D ou D après le (CTRL)-Q comme les éditeurs *WordStar* et *Borland* le feraient.

Pour savoir si (↑) ou (↓) a été pressée, vous devrez simplement contrôler la valeur de la variable globale *LCCommande* après l'appel à *LitChaine* (*LitMot* ou toute routine de lecture appelée réellement). Si *LCCommande* = *LCUtl0*, il s'agit d'une touche (↑) et si *LCCommande* = *LCUtl1*, il s'agit d'une touche (↓).

Supposons maintenant que vous vouliez désactiver la commande *LCIns* (Bascule du mode insertion/recouvrement). Pour cela, vous devrez réaffecter la touche (INS) (qui est normalement affectée à *LCIns*) à la commande 0 appelée *LCRien* dans TPEDIT.

```
{Affecte <Ins> à la commande nulle}
if not AjoutCommandeEdit(0, 1, $5200, $0000) then
   {Pas d'erreur possible ici} ;
```

Cette instruction, outre la désactivation de la commande *LCIns*, libérera de la place dans le tableau des touches pour ajouter des commandes utilisateur. Si vous voulez réactiver la commande, exécutez la même instruction avec *LCIns* à la place de 0 comme premier paramètre (Cmd).

Enfin modifions une affectation de touche existante. Par exemple, échangeons (HOME) par (CTRL)-(HOME) et (END) par (CTRL)-(END) :

```
{Affecte <Ctrl Home> à LChome}
if not AjoutCommandeEdit(LChome, 1, $7700, $0000) then
   {Pas d'erreur possible ici} ;
{Affecte <Ctrl End> à LCend}
if not AjoutCommandeEdit(LCend, 1, $7500, $0000) then
   {Pas d'erreur possible ici} ;
```

Lorsque *AjouCommandeEdit* parcourra le tableau des touches, elle constatera que (CTRL)-(HOME) est déjà affectée à *LCEffDebLg* (effacement depuis le début de la ligne jusqu'à la position du curseur) et l'affectera maintenant à *LCHome*. De la même façon, (CTRL)-(END) sera affectée à *LCEnd* plutôt qu'à *LCEffFinLg* (effacement de la position du curseur jusqu'à la fin de la ligne). Dans les deux cas, la place libre dans le tableau destiné aux ajouts sera inchangée.

La troisième et dernière méthode d'adaptation des touches est d'écrire un programme d'installation. Le sous-répertoire BONUS contient un programme de démonstration TPKEYS qui permet d'adapter les affectations des touches en fonction des besoins de vos utilisateurs. TPKEYS utilise l'interface clavier programmable (voir la documentation de TPKEYS pour plus de détails.

Ces trois méthode permettent d'adapter les unités en fonction de la souris. En consultant TPEDIT.PAS, vous verrez les déclarations suivantes :

```
{$IFDEF UseMouse}
3,    $00,   $EF,    LCsaisie        {bouton gauche = Enter}
3,    $00,   $EE,    LCquitte        {bouton droit  = ESC}
3,    $00,   $ED,    LCaide          {les deux à la fois    = Aide}
```

Ainsi pour adapter le rôle des boutons de la souris, vous n'avez besoin que des pseudo-codes de balayage retournés par la routine *SourisPressee* de TPSOURIS.

```
SourisGauche = $EF00;              {bouton gauche}
SourisDroit = $EE00;               {bouton droit}
SourisLes2 = $ED00;                {les 2 boutons}
{-------Souris possédant 3 boutons-------}
SourisCentral = $EC00;             {bouton central}
SourisGaucheCentral = $EB00;       {boutons gauche et central}
SourisDroitCentral = $EA00;        {boutons droit et central}
SourisLes3 = $E900;                {les trois boutons}
```

Avec ces données, vous pouvez adapter TPEDIT pour que la frappe des trois boutons de la souris corresponde à (CTRL)-R :

```
if not AjoutCommandeEdit(LCrestaure, 1, SourisLes3, $0000) then
   {erreur, table des touches pleine};
```

Encore une fois, mesurez bien les conséquences des modifications apportées à une unité donnée. Les boutons de la souris dans les cinq unités fournissent une interface utilisateur cohérente : le bouton gauche entraîne une sélection, le bouton droit est équivalent à (ESC) et les deux boutons affichent éventuellement un écran d'aide.

Interface de tâches d'arrière-plan

Les programmes d'arrière-plan (affichage de l'heure par exemple) sont parfois appréciés pendant l'attente de la frappe d'une touche. TPEDIT, TPENTRE, TPPICK, TPEDMAC, TPMENU et TPAIDE permettent de le faire facilement.

La première étape est l'exécution d'une fonction de la forme suivante (si le support de la souris est désactivé) :

```
uses TpCrt;
...
{$F+}
function LitTouche : Word;
begin
   while not KeyPressed do begin
      inline($CD/$28); {permet au TSR de prendre le contrôle}
      {** Exécution de la tâche de fond **}
   end;
   LitTouche := LitMotTouche;
end;
{$F-}
```

Si vous utilisez TPSOURIS dans votre programme, la fonction sera légèrement différente :

```
uses TpCrt, TpSouris;
...
{$F+}
function LitTouche : Word;
var
   I : Word;
begin
   I := $FFFF;
   repeat
      if KeyPressed then
         I := LitMotTouche
      else if SourisPressee then
         I := MotToucheSouris
         else begin
            inline($CD/$28); {permet au TSR de prendre le contrôle}
            {** Exécution de la tâche de fond **}
         end;
   until I <> $FFFF;
   LitTouche := I;
end;
{$F-}
```

Notez que la priorité est toujours donnée aux touches du clavier plutôt qu'aux boutons de la souris. Cela garantit que la frappe d'un bouton de la souris n'interrompra pas une macro clavier (toutes les unités qui comportent des interfaces de tâche d'arrière-plan, sauf TPEDMAC, supportent la souris).

La seconde étape est de faire en sorte que l'unité en question appelle cette routine lorsque cela est nécessaire pour lire une touche clavier. Dans chaque cas, cela se fait simplement en affectant l'adresse de votre routine à l'une des variables pointeurs suivantes :

```
PtrToucheEdit   {TPEDIT}
PtrToucheSaisie     {TPSAISIE}
PtrTouchePick   {TPPICK}
PtrToucheEdMac  {TPEDMAC}
PtrToucheAide   {TPAIDE}
PtrToucheMenu   {TPMENU}
```

Par exemple, pour dire à TPEDIT d'utiliser notre routine *LitTouche* :

```
PtrToucheEdit := @LitTouche
```

Puisque chaque unité a sa propre interface, vous pouvez utiliser une routine différente pour chaque unité ou utiliser la même dans chacune d'entre-elles. Si vous devez désactiver une ou plusieurs de vos routines, vous pouvez le faire simplement en initialisant la variable pointeur appropriée avec '*@LitMotTouche*' (si le support de la souris est désactivé) ou '*@LitToucheOuBouton*' (si le support de la souris est activé).

Quelques remarques sur les routines de lecture de touche méritent d'être notées. Premièrement, elles doivent être déclarées avec le modèle far ({$F+}). Deuxièmement, il est préférable d'appeler *KeyPressed* en début de boucle de polling plutôt qu'à la fin. Ceci évitera qu'une tâche d'arrière plan soit appelée s'il y a déjà une touche en attente. Troisièmement, l'instruction 'inline($CD/$28)' est quelquefois nécessaire pour lancer des programmes résidents pendant l'attente d'une touche clavier. Cette instruction génère l'interruption $28 qui garantit qu'un programme résident peut être activé en toute sécurité. Nous vous recommandons de toujours inclure cette instruction dans vos routines de lecture de touche (les programmes TSR sont les plus concernés).

Les règles de base pour une tâche d'arrière-plan sont simples : la tâche doit être rapide pour ne pas retarder les entrées au clavier, ou interruptible. Supposons par exemple que votre tâche d'arrière-plan mette à jour un ensemble de fenêtres et que la procédure *MiseAJourFen* modifie une fenêtre donnée. La procédure qui appelle cette routine ressemblera à ce qui suit :

```
procedure MetFenetreAJour;
   {Met les fenêtres à jour en attendant la frappe d'une touche}
const
   NumFen : Byte = 1;
var
   I : Byte;
begin
   {$IFDEF UseMouse}
   while not KeyPressed or SourisPressee do begin
   {$ELSE}
   while not KeyPressed do begin
   {$ENDIF}
      MiseAJourFen(NumFen);
         if NumFen = FenAMettreAJour then
            NumFen := 1
      else
         Inc(NumFen);
   end;
end;
```

Avant de mettre à jour le fenêtre suivante, il faut vérifier qu'il n'existe pas une touche en attente (ou qu'un bouton de la souris n'a pas été pressé). Dans ce cas, *MetFenetreAJour* s'arrête pour que la demande soit traitée. Sinon, la fenêtre est mise à jour, un compteur est incrémenté. Celui-ci sera remis à 1 lorsque toutes les fenêtres auront été mises à jour. Comme *NumFen* est une constante typée et non une variable locale, elle conserve sa valeur entre les appels, si bien que le prochain appel de *MetFenetreAJour* renverra le bon contexte. C'est le principe d'une routine interruptible : elle redonne rapidement le contrôle dès qu'une touche est pressée et reprend son exécution au prochain appel.

Cependant, rien n'indique que la valeur retournée est émise par *LitMotTouche* ou *MotToucheSouris*. Vous pourriez par exemple utiliser cette interface pour implémenter un système de macros simple. L'utilisateur pourra définir au plus dix chaînes de caractères enregistrées dans le tableau *ChaineMacros* qui seront associées aux touches de fonction (ALT)-(SHIFT). Si l'utilisateur presse (ALT)-(F1), vous retournerez '*Ord(ChaineMacros[1][1])*' plutôt que (ALT)-(F1). Au prochain appel de *LitTouche*, vous renverrez immédiatement '*Ord(Macro[1][2])*' et ainsi de suite jusqu'à ce que tous les caractères de *ChaineMacro[1]* soient renvoyés. Ensuite, vous pourrez revenir à *LitMotTouche* normalement.

Bien que ce système de macros soit moins simple que les macros clavier de TPMACRO, il offre certains avantages notamment dans les programmes où la saisie de données est importante.

Interface d'aide adaptée au contexte

Pour faciliter l'utilisation de TPAIDE, les unités TPEDIT, TPSAISIE, TPPICK, TPMENU et TPEDMAC définissent toutes une commande d'aide (affectée à (F1) par défaut) et une interface qui vous permet de mentionner une routine qui sera appelée lorsque cette commande est exécutée.

Les étapes de la mise en oeuvre de cette routine sont identique à celles mises en oeuvre pour contrôler des entrées clavier. Avant de décrire ces étapes, prenez le temps de lire la documentation de l'unité TPAIDE si ce n'est déjà fait. Nous supposons dans ce qui suit que vous avez acquis les connaissances de base sur le rôle de cette unité et sur son mode d'emploi.

La première étape consiste à écrire une routine qui traitera les commandes d'aide. Cette routine est une procédure globale déclarée avec le modèle FAR qui comporte trois paramètres :

```
{$F+}
procedure RoutineAide(CodeUnite:Byte; PtrId:Pointer; IndexAide:Word);
begin
   case CodeUnite of
      AidePourPick,        {TPPICK}
      AidePourModif,       {TPEDIT}
      AidePourMenu,        {TPMENU}
      AidePourSaisie,      {TPSAISIE}
      AidePourEditMacro :  {TPEDMAC}
      {Ajuster si nécessaire IndexAide, basé sur CodeUnite et PtrId} ;
   end;

   {Taitement de la demande d'aide}
   if not AffAide(MonPtrAide, IndexAide) then
      {Gestion de l'erreur};
end;
{$F-}
```

Le premier paramètre, le code de l'unité, sera en général égal à l'une des constantes de l'exemple. Ces constantes sont définies dans TPCRT (puisque les cinq autres unités en dépendent). Voici la liste complète :

AidePourUtl	=	00;	{inutilisé}
AidePourPick	=	01;	{TPPICK}
AidePourModif	=	02;	{TPEDIT}
AidePourMenu	=	03;	{TPMENU}
AidePourSaisie	=	04;	{TPSAISIE}
AidePourEditMacro	=	05;	{TPEDMAC}
AidePourXXXX1	=	06;	{Réservé}
AidePourXXXX2	=	07;	{Réservé}
AidePourXXXX3	=	08;	{Réservé}
AidePourXXXX4	=	09;	{Réservé}
AidePourXXXX5	=	10;	{Réservé}

Lorsque vous devrez appeler votre routine d'aide directement, vous passerez *AidePourUtl* ou une valeur supérieure à 10 comme *CodeUnite*. Les valeurs comprises entre 6 et 10 sont réservées à *Turbo Professionnel*.

Le second paramètre *PtrID* vaut nil en général. Si *CodeUnite* vaut TPPICK ou TPMENU, *PtrID* contiendra une adresse non significative. Lors de l'appel d'une routine d'aide, TPPICK passe toujours l'adresse de la routine que vous (ou TPDIR) spécifiez comme paramètre *FoncChaine* pour *FenPick*. TPMENU passe l'adresse du menu courant. Dans les programmes qui contiennent plusieurs menus ou plusieurs appels à *FenPick*, ces adresses peuvent servir à déterminer l'emplacement où vous étiez dans le programme lorsque la demande d'aide a été activée.

Le troisième paramètre, *IndexAide*, indique également l'emplacement du programme où vous vous trouvez. Sa valeur varie énormément d'une unité à une autre :

- TPPICK : transmet le numéro de l'élément mis en valeur dans la liste pick
- TPEDIT : transmet la valeur courante de la variable globale *EditeIndexAide* (vous devez l'initialiser avant d'appeler *LitChaine* ou toute autre routine de lecture)
- TPMENU : transmet la touche menu de l'élément courant mis en valeur (une bonne raison pour que la touche menu de chaque élément soit unique)
- TPSAISIE : transmet l'index d'aide que vous avez affecté à la rubrique en cours de modification (voir chapitre 3).
- TPEDMAC : transmet toujours zéro si l'utilisateur appelle les écrans d'aide simplement avec l'éditeur de macros. Votre routine d'aide doit fournir l'index correct dans le fichier d'aide lorsque *CodeUnite* est égal *AidePourEditMacro*.

Etant donné la diversité des rôles de *PtrID* et *IndexAide*, l'objet de *CodeUnite* est maintenant évident : permettre à la routine d'aide de déterminer, étant donné *IndexAide* et *PtrID*, la valeur du thème à transmettre à la routine *AffAide* comme paramètre.

La deuxième étape est d'indiquer à l'unité en question quelle routine elle doit appeler lorsque la commande d'aide est exécutée. A nouveau, il faudra attribuer l'adresse de votre routine à l'une des variables pointeur qui suivent :

```
PtrAideEdit     {TPEDIT}
PtrAideSaisie   {TPSAISIE}
PtrAidePick     {TPPICK}
PtrAideEdMac    {TPEDMAC}
PtrAideMenu     {TPMENU}
```

La valeur par défaut de ces variables est nil, ce qui dans tous les cas sauf un conduit à ignorer les commandes d'aide. Dans le cas de TPEDMAC, la frappe de (F1) en mode commande insérera cette touche dans la macro en cours de modification si *PtrAideEdMac* vaut nil, sinon, elle appellera la routine d'aide qu'elle pointe (la touche sera toujours insérée quelque soit la valeur de *PtrAideEdMac*).

Ces deux étapes résolues, vous n'avez plus qu'à écrire le texte qui sera affiché sur vos écrans d'aide et à exécuter l'utilitaire *CreeAide*.

Les paragraphes qui précèdent contiennent beaucoup d'informations que vous devrez probablement relire plusieurs fois. Mais avant de les lire une seconde fois, il est urgent que vous consultiez SAISIE.PAS et SAISIE.TXT (fichier ASCII utilisé par *CreeAide* pour créer SAISIE.HLP) et que vous exécutiez SAISIE.EXE afin d'utiliser son système d'aide. En observant ce système d'aide, vous comprendrez mieux les informations précédentes lors de votre relecture.

Utilisation des "overlays" avec Turbo Professionnel

Indiscutablement, l'une des améliorations les plus importantes de *Turbo Pascal* 5.0 est le retour aux "overlays". Notre expérience des produits *Turbo Extender, Overlay Manager* et autres, nous a prouvé que de nombreux programmeur en *Turbo Pascal* géraient de grandes applications, et avaient besoin des "overlays" moins par choix que par nécessité. Si vous appartenez à ces milliers d'utilisateurs, vous apprendrez avec plaisir que les unités de *Turbo Professionnel* fonctionnent correctement avec des programmes overlays.

Le première et la plus importante des étapes afin de supporter le nouveau gestionnaire "d'overlays" a été de modifier ou d'ajouter des directives de compilation dans toutes les unités du logiciel. Cela signifie que nous avons placé la directive de compilation {$F+} dans les unités où cela était nécessaire (celles qui doivent appeler des routines utilisateur qui doivent être "overlays") ou {$F-} dans celles qui n'en ont pas besoin et placé la directive {$O-} dans les autres unités (en général, parce que l'unité contient un gestionnaire d'interruption ou une routine qui pourrait être appelée).

Dans de nombreux cas, nous avons déterminé les unités pouvant être mises en "overlays" sans risque et nous avons spécifié les directives {$F+} et {$O+}. Ces unités sont dans l'ordre alphabétique :

```
TPASCIIZ
TPBCD
TPDATE
TPDIR
TPDOS
TPTBMEXP
TPEDIT
TPSAISIE
TPMEMEXT
TPAIDE
TPEDMAC
TPMEMCHK
TPMENU
TPPICK
TPTBMRAM
TPTRI
TPTBMVIR
TPFENETR
```

Avant de placer des unités en overlays, étudiez soigneusement la liste des dépendances entre unités (voir annexe A). Dans certains cas, vous éliminerez certaines unités pour des raisons de performances. Par exemple, faites bien attention avant d'utiliser des overlays avec TPFENETR et TPMENU, puisque TPMENU doit accéder fréquemment aux routines de TPFENETR.

La deuxième étape a consisté à modifier l'unité TPTSR pour vous permettre d'écrire des programmes résidents en mémoire qui contiennent des overlays (nous épargnerons les détails des modifications internes que nous avons apportées). Cela n'est pas aussi difficile que vous pouvez le penser, le programme de démonstration qui suit en est un exemple (il est basé sur le programme OVRDEMO.PAS de Borland) :

```
{$V-,S-,I-,R-,F+,O+}
{$M 4096, 0, 650000}
program OvrDemo;

uses Overlay, Dos ,TpDos, TpTsr, TpCrt, DemoOvr1, DemoOvr2;

{$O DemoOvr1}
{$O DemoOvr2}

const
   ToucheTab = $080F; {ALT-TAB}
   Compte : Word = 0;
var
   NomOvr : string[64];
```

```
procedure ProcPop(var Regs : Registers);
   {Routine activée si la séquence ALT-TAB est pressée}
begin
   if not LibereEMS then
      if not FichExiste(NomOvr) then
         {Erreur : pas d'appel de routines en overlay} ;
   {incrémentation du compteur}
   Inc(Compte);
   WriteLn(Routine popup. Compteur = ', Compte,'.');

   {appel de routines d'unités en overlay}
   Routine1;
   Routine2;

   {désactive le programme après 10 popups}
   if Compte >= 10 then
      if DesactiveResid then {Succès} ;
end;

begin
   {localisation du fichier overlay}
   if not ExisteSurChemin('DEMOOVR.OVR',NomOvr) then begin
      WriteLn('DEMOOVR.OVR introuvable.');
      Halt(1);
   end;

   {ouverture du fichier overlay}
   OvrInit(NomOvr);
   if OvrResult <> 0 then begin
      WriteLn('Erreur overlay : ', OvrResult);
      Halt(1);
   end;

   {place l'overlay en EMS si possible}
   OvrInitEms;
   Case OvrResult of
      ovrIOError : WriteLn('Erreur E/S fichier overlay);
      ovrNoEMSDriver : WriteLn('Driver EMS non installé');
      ovrNoEMSMemory : WriteLn('Mémoire EMS insuffisante');
   end;

   {si la demande d'EMS aboutit, initialise le drapeau LibereEMS}
   LibereEMS := (OvrResult = 0);

   {ALT TAB activera cette popup}
   if not DefinitPop(ToucheTab, @ProcPop, Ptr(SSeg, SPtr), True) then
      WriteLn('Impossible de créer la routine popup');
   else begin
      WriteLn('Popup chargée, tapez ALT-TAB pour l''activer.');
      PopupsOn;
      if not TermineEtResteResident(ParagraphesAConserver, 0) then
         WriteLn('Impossible d''installer la partie résidente');
   end;
end.
```

Bien que ce programme contrôle peu les erreurs, il démontre deux points importants. Premièrement, vous devez être absolument certain que votre programme accède à son fichier overlay lorsque vous l'activerez. Le meilleur moyen est de connaître le nom du chemin complet pour le fichier overlay lorsque votre programme commence, et d'utiliser la routine *FichExiste* de TPDOS pour contrôler son existence lorsque vous lancerez l'exécution du programme. Si le fichier overlay n'est pas trouvé, vous afficherez un message d'erreur et sortirez de votre programme popup sans appeler de routine overlay. Remarquez que le contrôle du fichier overlay n'est pas nécessaire si la mémoire EMS est utilisée pour enregistrer les overlays.

Deuxièmement, si vous utilisez la routine *OvrInitEMS* pour enregistrer le contenu de votre fichier overlay en mémoire EMS, vous devrez initialiser la variable *LibereEMS* de l'unité TPTSR. Cela garantit que la mémoire EMS allouée par le gestionnaire d'overlays sera libérée lorsque vous désactiver votre programme TSR. Remarquez qu'il n'est en général pas nécessaire de sauvegarder et de restaurer le contexte du gestionnaire de mémoire étendue puisque le gestionnaire d'overlays de *Turbo Pascal* 5.0 le fait à votre place.

Si vous suivez ces deux conseils, vous n'aurez aucune difficulté à écrire un programme TSR qui utilise le système "d'overlays" de *Turbo Pascal* 5.0.

Index

A

AbsBcd 11.4
ActiveAideSouris 2.199
ActiveAttenteEvenement 2.234
ActiveEcranV 2.86
ActiveEditSouris 2.175
ActiveElemMenu 2.125
ActiveFen 2.88
ActiveFenDisj 2.89
ActiveIndexAide 2.201
ActiveMenuSouris 2.126
ActivePickSouris 2.272
ActiveSaisieSouris 3.34
AdapteCouleursFenCompactee 2.16
AddBcd 11.4
AdresseMacro 9.5
AffAide 2.202
AffAidePreced 2.205
AffecteNomMacro 9.6
AffFenCompactee 2.17
AffFenCompacteeEn 2.17
AfficheFen 2.90
AffSouris 2.235
AideEcran 2.206
AjoutBackSlash 4.5
AjoutChampArray 3.35
AjoutChampBcd 3.36
AjoutChampBoolean 3.38
AjoutChampByte 3.39
AjoutChampChar 3.40
AjoutChampChnDate 3.41
AjoutChampChoix 3.42
AjoutChampComp 3.44
AjoutChampDate 3.45
AjoutChampDbl 3.46
AjoutChampEmboite 3.47
AjoutChampExt 3.48
AjoutChampHeure 3.49
AjoutChampInt 3.50
AjoutChampLong 3.50
AjoutChampOuiNon 3.51
AjoutChampReal 3.51
AjoutChampSgl 3.52
AjoutChampShort 3.52
AjoutChampString 3.53
AjoutChampUtl 3.55
AjoutChampWord 3.56
AjoutFenBiblio 2.18
AlloueEcran 2.295
AlloueMacro 9.7
AlloueMemExt 8.21
AllouePagesEms 8.5
AllouePile 6.8
AlloueT 10.5, 10.22, 10.35
AnalyseChemin 5.4
AnnuleImpFich 5.5
AnnuleImpTousFichs 5.6
AppelAncISR 12.8
AppelFar 12.9
AppelNear 12.10
ApureT 10.37
ArcTanBcd 11.5
ArrangeMots 4.6
AscCaracChn 4.64
AscCentre 4.64
AscCentreCar 4.65
AscComplete 4.66
AscCompleteCar 4.67
AscCompleteGauche 4.68
AscCompleteGaucheCar 4.69
AscDepuisTas 4.70
AscEff 4.71
AscEffDebut 4.72
AscEffFin 4.73
AscEnChn 4.74
AscMajusc 4.75
AscMinusc 4.76
AscSurTas 4.77
AssignCrt 2.296
AsseciePageEms 8.6
Aujourdhui 3.103
AujourdhuiEnChaine 3.103

B

BcdEnReel 11.5
BlancsInterdits 3.57
BoutonSourisPresse 2.236
BoutonSourisRelache 2.237

C

CacheSouris 2.238
CadreDroite 2.19
CadreFen 2.20
CaracMacro 9.8
Centre 4.7
CentreCar 4.7
CentreVite 2.21
ChaineCarac 4.8
ChaineDateEnDate 3.104
ChaineDateEnJMA 3.104
ChaineDuTas 4.9
ChaineHeureCourante 3.105
ChaineHeureEnHms 3.105
ChainePourTriEnDate 3.106
ChainePourTriEnHeure 3.107
ChaineSurTas 4.10
ChainInt 6.9
ChangeAttr 2.22
ChangeAttrFen 2.23
ChangeHotKey 6.34
ChangeProtection 3.58
ChargeT 10.6, 10.23, 10.38
ChargeTV 10.7
ChercheAide 2.208
ChercheElemMenu 2.128
ChnBcd 11.6
ChnEnAsc 4.78
ChnEnEntier 4.11
ChnEnLongEntier 4.12
ChnEnMot 4.13
ChnEnReel 4.14
ChnExpBcd 11.7
ChnMajusc 4.15
ChnMinusc 4.15
ChoixMenu 2.129
ChoixPick 2.274
CodesIdentFichsOuverts 5.7
ColsActuelles 2.296
ColsEcranV 2.91
CompacteBiblio 2.24
CompacteFen 2.25
CompareAsc 4.78
CompareAscMajusc 4.79
CompareChaine 4.16
CompareChnMajusc 4.17
CompareEnsLettres 4.18
CompareStruct 4.20
CompBiblioMenu 2.132
Complete 4.21
CompleteCar 4.22
CompleteGauche 4.23
CompleteGaucheCar 4.24
CompteMots 4.25
ConcatAsc 4.80
ConcatChn 4.81
ConstruitTableBM 4.26
ContrlClavier 2.26
ContrlEtatMenu 2.133
CoordXAbs 2.27
CoordXY 2.28
CoordYAbs 2.29
CopieAsc 4.82
CopieEcran 2.297
CopieFen 2.299
CorrigeNomChemin 4.27
CosBcd 11.8
CouleurCouleurs 2.301
CouleursMono 2.302
CreeBiblio 2.30
CreeEcranV 2.92
CreeEnsLettres 4.28
CursBloc 2.31
CursCache 2.31
CursDebFin 2.32
CursNormal 2.33
CursPlat 2.33
CursSourisBloc 2.238
CursSourisCache 2.239
CursSourisHard 2.240
CursSourisNormal 2.241
CursSourisPlat 2.241
CursSourisSoft 2.242

D

DateEnChaineDate 3.107
DateEnChainePourTri 3.108
DateEnJMA 3.109
DateInternationale 3.110
DealloueAsc 4.83
DealloueChaine 4.29
DealloueCodeIdentEms 8.8
DealloueEcran 2.303

DeallouеFen 2.93
DeallouеMacro 9.8
DeallouеMemExt 8.22
DeallouеMenu 2.134
DeallouеPile 6.10
DeallouеT 10.8, 10.24, 10.39
DefinitEcran 2.304
DefinitMacro 9.9
DefinitPop 6.35
DefinitProcPop 6.36
DeplaceEcran 2.34
DeplaceEcrVDansFen 2.94
DeplaceFen 2.96
DeplaceFenDansEcrV 2.97
DeplaceFenHoriz 2.305
DeplaceFenVerti 2.306
DeplaceMemExt 8.23
DerouleFen 2.98
DerouleFenBas 2.35
DerouleFenHaut 2.36
DesactiveAideSouris 2.210
DesactiveAttenteEvenement 2.243
DesactiveEcranV 2.99
DesactiveEditSouris 2.175
DesactiveElemMenu 2.135
DesactiveIndexAide 2.211
DesactiveMenuSouris 2.137
DesactivePickSouris 2.276
DesactivePop 6.37
DesactiveResid 6.38
DesactiveSaisieSouris 3.59
DessineChamp 3.59
DessineEcranEdit 3.60
DessineMenuPile 2.138
DiffDate 3.111
DiffDateEnJourSec 3.112
DifferentBcd 11.9
DiffHms 3.113
DivBcd 11.9

E

EchangeMots 12.11
EchangeOctets 12.11
EchangeStructs 12.12
EchPileEtAppel 6.11
EchPileEtAppelNear 6.13
EcranEdit 3.61
EcritAsc 4.84
EcritAttr 2.37
EcritAttrFen 2.37
EcritBiblioMenu 2.139
EcritCentre 2.307
EcritEcranV 2.100
EcritElemT 10.9, 10.25, 10.40
EcritFacile 2.38
EcritFacileFen 2.39
EcritFenCompactee 2.40
EcritFichMacro 9.9
EcritMemVite 2.308
EcritRapide 2.309
EcritSecteursDisque 5.8
EcritVite 2.41
EcritViteEtCoupe 2.101
EcritViteFen 2.42
EcritVitePile 2.102
EditeChaine 3.63
EffaceEcranV 2.103
EffaceEntFen 3.65
EffaceFenSup 2.104
EffaceMacros 9.10
EffaceMenu 2.141
EffaceMenuPile 2.142
EffaceSousMenuActuel 2.143
EffBlancs 4.30
EffEspDeb 4.31
EffEspFin 4.32
EgalBcd 11.10
ElemMenu 2.145
EmsPresent 8.9
EmuleInt 6.14
EnleveMacros 9.10
EnleveModule 6.39
EnregistreCoordsFen 2.43
EnregistreCoordsFen 2.310
EnregistrementMacroOff 9.11
EnregistrementMacroOn 9.11
EnregistreT 10.11, 10.27, 10.42
EnregistreTV 10.12
EnvoiFinInt 6.15
Erreur8087 6.53
ErreurT 10.13, 10.28, 10.43
EtatMenu 2.147
Exceptions8087 6.54
ExecDos 5.10
ExisteSurChemin 5.13
ExpBcd 11.11
ExtraitMot 4.33

F

FenEstActive 2.105
FenPick 2.277
FenSouris 2.244
FenSourisPleinEcran 2.245
FermeAide 2.212
FermeBiblio 2.44
FichExiste 5.14
FinEtatImp 5.15
FixeCouleur 2.45
Fonte8x8Sel 2.46
ForceSuffixe 4.34
Formate 4.35
Formate 11.11
FracBcd 11.12

H

HauteurMenu 2.148
HercPresent 2.47
HeureCourante 3.114
HeureEnChaineAmPm 3.115
HeureEnChaineHeure 3.115
HeureEnChainePourTri 3.116
HeureEnHms 3.117
HeureInternationale 3.118
HmsEnSecondes 3.119
HorlogeMs 5.15

I

IdentEstConsole 5.16
IncDate 3.119
IncHms 3.120
IndicateurExiste 12.13
IndicDosActif 6.39
IndicErrDos 6.40
IndicsClavier 2.48
InferieurBcd 11.12
InferieurEgalBcd 11.13
InitAttrChaine 3.65
InitAttrChamp 3.66
InitAttrCtrl 3.67
InitAttrMess 3.68
InitAutoNumLock 3.69
InitAvanceAuto 3.70
InitBipSurErreur 3.71
InitBordCrt 2.49
InitCadreInactif 2.106
InitCaracComplement 3.72
InitCaracMotDePasse 3.73
InitCaracsCadre 2.50
InitCarFond 2.311
InitClign 2.51
InitCloche 3.73
InitCurseurEnFin 3.74
InitDelimiteurs 3.75
InitDta 5.17
InitEffPremCarac 3.76
InitElimineBlancs 3.77
InitEnregEs 3.78
InitEvenementsSouris 2.246
InitFen 2.107
InitFenDisjSup 2.109
InitFenEnt 3.79
InitFenSup 2.110
InitForceMode 3.80
InitHorlogePop 6.41
InitIndicateur 12.14
InitInsereEtPousse 3.81
InitJustifDroite 3.82
InitMacros 9.12
InitMenu 2.150
InitMenuDelai 2.151
InitModeEnroule 3.83
InitModeHerc 2.52
InitModeMenu 2.152
InitModeMotDePasse 3.84
InitNomElemMenu 2.153
InitNumerique 3.84
InitObligatoire 3.85
InitPageActive 2.312
InitPageSouris 2.247
InitPageVisible 2.53
InitPops 6.42
InitPosAide 2.213
InitProtection 3.86
InitPtrApresEdit 3.87
InitPtrAvantEdit 3.88
InitPtrErreur 3.89
InitRatioMickeyEnPixel 2.248
InitSaut 12.15
InitSortieSurSecondClic 3.90
InitSouris 2.249
InitT 10.15, 10.30, 10.45
InitTailleCurs 2.54
InitToucheSelMenu 2.155
InitVecteur 6.16
InitVerif 5.18
InitVideo 2.313

InsereAsc 4.85
InsereChn 4.86
InstalleModule 6.43
Int24Off 6.59
Int24On 6.59
IntBcd 11.13
InterruptsOff 6.17
InterruptsOn 6.17

J

JMAEnChaineDate 3.120
JMAEnDate 3.121
JourSemaine 3.121

L

LanceImpFich 5.19
LanceMacro 9.13
LanceMacroParTouche 9.14
LanceSousNiveau 2.156
LargeurMenu 2.157
LecteurParDefaut 5.20
LibereEcranEdit 3.91
LibereEcranV 2.112
LibereFenCompactee 2.55
LibereVideo 2.314
LigneDebutCurs 2.55
LigneFinCurs 2.56
LignesActuelles 2.315
LignesEcranV 2.113
ListeAide 2.214
LitAsc 4.87
LitAttr 2.57
LitAttrCurs 2.58
LitAttrFen 2.58
LitBiblioMenu 2.158
LitCaracSousCurs 2.59
LitCaractere 2.176
LitChaine 2.177
LitChn 4.93
LitDta 5.21
LitEcranV 2.114
LitElemT 10.16, 10.31, 10.46
LitEntier 2.178
LitEnvironnement 5.22
LitEtatCurs 2.60
LitEtatImp 5.23
LitFenBiblio 2.61
LitFenCompactee 2.62
LitFichMacro 9.15
LitInfoDisque 5.24
LitLigneVite 2.315
LitLongEntier 2.179
LitModeFich 5.25
LitModeVideo 2.63
LitMot 2.181
LitMotTouche 2.64
LitNomElemMenu 2.159
LitNomFic 2.286
LitPageSouris 2.250
LitReel 2.182
LitSecteursDisque 5.26
LitTotalMickey 2.251
LitToucheOuBouton 2.252
LitVite 2.65
LitViteFen 2.65
LnBcd 11.14
LongAsc 4.88
LongEnChn 4.37
LongEntEnBcd 11.14
LongEntierBin 4.38
LongEntierHex 4.38
LongEntierOctal 4.39

M

MacroMajuscule 4.39
MacroMinuscule 4.39
MacrosOff 9.16
MacrosOn 9.17
MarqueMemExt 8.24
MemExtDisp 8.25
Minuscule 4.41
ModeGraphHerc 2.66
ModeMenu 2.160
ModifBloc 5.28
ModifTouches 9.21
ModulePresent 6.44
MonnaieInternationale 3.92
MotBin 4.42
MotHex 4.43
MotMax 12.16
MotMin 12.16
MotOctal 4.43
MotPoidsFaible 12.17
MotPoidsFort 12.17
MotToucheSouris 2.253
MultBcd 11.15

N

NbCodesIdentEmsActifs 8.10
NbLecteurs 5.28
NbMacro 9.17
NomCheminComplet 4.44
NomCheminSeul 4.45
NomFichSeul 4.45
Normalise 12.18
NouvMenu 2.161

O

OctetBin 4.46
OctetHex 4.46
OctetOctal 4.47
OuiOuNon 2.184
OuvreBiblio 2.67
OuvreFicAide 2.216
OuvreMemAide 2.218
OuvreUniteStd 5.29

P

PagesEmsDisp 8.11
PagesEmsPrises 8.12
ParagraphesAConserver 6.45
PermuteMot 12.18
PlaceChaine 2.68
PlaceElement 7.5
PlaceEnXYAbs 2.69
PlaceTab 4.48
PlaceTouche 2.70
PopupsOff 6.46
PopupsOn 6.47
PosAsc 4.89
PosChn 4.90
PosTexte 5.29
PrendElement 7.6
Presence8087 6.55
PrimitiveAlloueT 10.18, 10.48
PrintPresent 5.30
PtrCadrePageEms 8.13
PtrElemMenu 2.163
PtrEnEtendu 8.26
PtrHex 4.49
PtrModuleParNom 6.48

R

RecadreFen 2.115
RechBMSansMajMin 4.50
Recherche 4.52
RechercheBM 4.53
RechercheIndexMacro 9.18
RechercheSansMajMin 4.55
RechTexte 5.31
ReelEnBcd 11.16
ReelEnChn 4.56
ReinitEcran 2.316
ReinitVideo 2.71
RemplitCaracAttr 2.317
RemplitCaracVideo 2.318
RemplitFen 2.319
RemplitFenPick 2.280
RemplitMot 12.19
RemplitStruct 12.20
RemplitVite 2.72
RemplitViteFen 2.72
RendMemExt 8.27
RepriseErreur 12.5
Restaure8087 6.56
RestaureContexteEms 8.15
RestaureCoordsFen 2.73
RestaureCoordsFen 2.319
RestaureEtatCurs 2.73
RestaureEtatSouris 2.254
RestaureFen 2.74
RestaureTousVecteurs 6.18
RestaureVecteur 6.19
RestaureVecteursClavier 5.40
ResultInt24 6.48; 6.60
RetourInt 6.20
RetourneChn 4.93
RetraceMenu 2.164
RoundBcd 11.17

S

SautAncIsr 12.21
SautLong 12.22
Sauve8087 6.57
SauveContexteEms 8.16
SauveEtatSouris 2.255
SauveFen 2.75
SelectEcran 2.320
SelectElemMenu 2.165
SelectLecteur 5.32
SelFonte8X8 2.76

SinBcd 11.18
SonneCloche 3.93
Soundex 4.57
SourisDansFen 2.256
SourisEnXY 2.257
SourisPressee 2.257
SourisX 2.258
SourisXY 2.259
SourisY 2.260
SousMenu 2.166
SoustraitBcd 11.18
SqrBcd 11.19
SqrtBcd 11.19
StatutEmsOK 8.17
SuffixeParDefaut 4.58
SuffixeSeul 4.59
SuperieurBcd 11.20
SuperieurEgalBcd 11.20
SupprimeAsc 4.91
SupprimeFenBiblio 2.77
SupprimePops 6.49
SupprimeProcPop 6.49
SupTab 4.60

T

TailleBuffSouris 2.261
TailleFichTexte 5.32
TermineEtResteResident 6.50
TesteModeHerc 2.78
TexteRapide 2.78
TexteRapideFen 2.79
TotalMemExt 8.28
TotalPagesEms 8.18
ToucheEnChaine 9.19
Tri 7.7
TruncBcd 11.21

U

UpCase 4.61

V

ValBcd 11.22
ValideSousChamps 3.94
VerifieDate 3.122
VerifOn 5.33
VersionDos 5.33
VersionEms 8.19
VertVite 2.79
VertViteFen 2.80
VideBufTexte 5.34
VideOuRempli 3.95

Z

ZeroIndicateur 12.23

Reproduit par INSTAPRINT S.A.
1-2-3, levée de la Loire – LA RICHE – B.P. 5927 – 37059 TOURS Cedex
Tél. 47 38 16 04

Dépôt légal 1er trimestre 1989